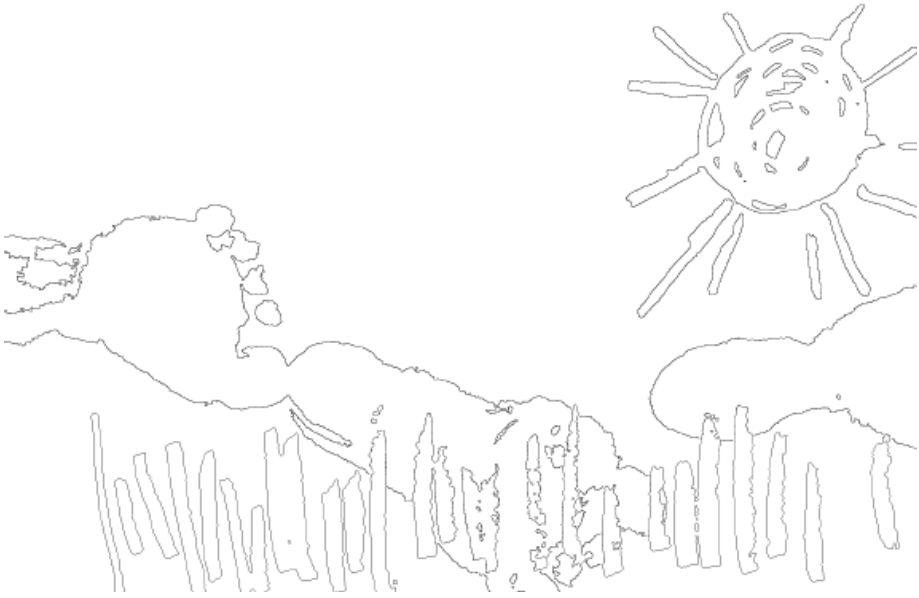
Guida ad Inform per Principianti

Roger Firth e Sonja Kesserich



Terza Edizione

Prefazione di Graham Nelson

Guida ad Inform per Principianti

Titolo originale: The Inform Beginner's Guide

Autori: Roger Firth e Sonja Kesserich

Copertina: *First Steps* – acquarello e pastello su carta, 2002 – Harry Firth (2000—)

Versione italiana:

Hanno collaborato alla traduzione italiana dell'Inform Beginner's Guide: Marco Falcinelli, Daniele A. Gewurz, Paolo Lucchesi, Paolo Vece e Giulio Veneziani.

La traduzione e l'adattamento dei giochi di esempio è stata curata da Paolo Lucchesi.

La revisione dei testi e dei giochi, l'adattamento, l'impaginazione, e l'aggiornamento alla terza edizione è stata curata da Marco Falcinelli.

La correzione delle bozze sono ad opera di Vincenzo Scarpa e Paolo Vece.

Prima Edizione: Aprile 2002

Seconda Edizione (con poche revisioni): Agosto 2002

Seconda Edizione (Italiano): Gennaio 2004

Terza Edizione (allineamento ad Inform 6.30, ulteriori revisioni): Agosto 2004

Terza Edizione (Italiano): Dicembre 2006

Sommario

Prefazione di Graham Nelson

Sarebbe immodesto da parte mia, penso, paragonarmi a Charles Bourbaki (1816-97), eroe francese della guerra di Crimea e illustre stratega, un uomo a cui fu offerto – come ricompensa – niente meno che il trono di Grecia (che egli rifiutò). Non è però fuori luogo dire qualche parola sul suo parente immaginario Nicholas, il più caparbio, lugubre, interminabilmente minuzioso e pesante autore di manuali che abbia mai enunciato un teorema. Un po' come Hollywood attribuisce i film dei quali nessuno vuole assumersi la responsabilità al regista "Alan Smithee" (che ha oramai una filmografia sostanziosa e a cui viene persino dedicata occasionalmente una retrospettiva), così in matematica molti piccoli risultati vengono dichiarati opera di Nicholas Bourbaki. Si narrano varie storie sulla nascita di Bourbaki, sotto il cui nome alcuni giovani matematici parigini si sono riuniti a partire dal 1935 per scrivere trattazioni generali di interi campi dell'algebra. Si noti che le sue iniziali sono NB. Alcuni dicono che "Bourbaki" era una spiritosaggine interna all'École Normale Supérieure (proprio come "zork" e "foobar" lo erano per il MIT), che risaliva a uno scherzo nel 1880, quando un allievo riuscì a fingere di essere un certo "Generale Claude Bourbaki" in visita. Si dice anche che il vero generale era famoso per essere in grado, durante le manovre, di mangiare *qualsiasi* cosa, se ce n'era bisogno: gallette stantie, rape crude, il suo cavallo, il fieno del suo cavallo, il sacco per il foraggio in cui era contenuto il fieno, proprio come Nicholas Bourbaki avrebbe mangiato tutto quello che c'era da mangiare nella teoria dell'algebra, per quanto duro da masticare o da mandar giù. Per dare a Cesare quel che è di Cesare, i quaranta volumi di Bourbaki sono alquanto utili. O piuttosto, non lo sono, ma fa piacere sapere che ci sono.

È stato leggendo il libro che avete in mano che mi sono reso conto della malinconica verità: che il mio volume su Inform, il *Designer's Manual*, è un Bourbaki. Deve trattare ogni singolo argomento, dagli accenti islandesi al linguaggio assembler, dalle pseudoazioni, per non parlare delle pseudopseudoazioni, a come si fa a raggruppare oggetti quasi identici, ma non del tutto, come tessere dello Scarabeo: tutti argomenti di cui un programmatore che si dedica con passione a Inform potrebbe avere bisogno una volta in tutta la vita, o anche no. Naturalmente, gli argomenti fondamentali vi compaiono di continuo, specialmente nei capitoli II e III, ma nonostante le mie migliori intenzioni si tratta di un modo comodo per cominciare solo se uno ci si fa strada come attraversando un fiume di pietra in pietra. Questo libro, d'altro canto, è un'introduzione che vi accompagna passo passo, che tratta le basi approfonditamente e poco per volta. Laddove il *Designer's Manual* cerca di non tornare mai sui propri passi, così che per esempio c'è solo un paragrafo sulle locazioni, *Guida Inform per Principianti* procede attraverso tre avventure intere, dando tre opportunità di rivedere ogni argomento, approfondendo e spiegando di più ogni volta.

Vorrei dire che la mia prima reazione, quando gli autori mi hanno mandato inaspettatamente le prime bozze, sia stata di gioire per l'approccio lucido, pulito e improntato al buon senso. In realtà, però, questa è stata solo la mia terza reazione, mentre la prima è stata di gelosia (la fate facile voi, che non dovete documentare come il parser calcola gli insiemi GNA per i predicati nominali) e la seconda di risentimento (*avete lanciato sul mio libro l'incantesimo gizbru: trasformare un oggetto pericoloso in uno inoffensivo*). Ma in realtà, non c'è maggior complimento che un autore possa ricevere che il fatto che qualcun altro decida di scrivere un libro sul suo lavoro, e così ringrazio Roger e Sonja per il loro gesto, nonché per l'ottimo lavoro che hanno fatto. Ho parlato abbastanza di me, considerando che Inform appartiene a tutti quelli che lo usano, alle centinaia di seri autori di narrativa interattiva che lo trovano utile, e che per quasi un decennio è stato un'impresa collettiva. Oggi nessuno ricorda più chi suggerì che cosa. Le sue regole per descrivere il world model oramai ricordano un patchwork del New England, a cui ogni casa del villaggio contribuisce tessendone un quadrato. Mentre leggete questo libro, tenete a mente che una di queste coperte non è mai conclusa e ha sempre uno spazio per il quadrato di un vicino appena arrivato.

St Anne's College

Università di Oxford

Aprile 2002

Introduzione

Le avventure testuali, note anche come narrativa interattiva (Interactive Fiction, IF), furono giochi per computer molto popolari negli anni Ottanta. Con l'evoluzione della tecnologia dei calcolatori svanirono progressivamente dal mercato, non potendo competere con i giochi grafici sempre più sofisticati; ma la narrativa interattiva era tutt'altro che morta. La crescita di Internet e dei gruppi di discussione Usenet (i newsgroup) ha avvicinato appassionati che avevano molto da dire. Sviluppando sistemi e strumenti di programmazione, organizzando concorsi, scrivendo recensioni e redigendo manifesti, questi appassionati hanno dato l'avvio a una rinascita che ha prodotto molte opere di grande qualità, comprese alcune che secondo molti hanno superato i migliori titoli commerciali degli anni '80.

Quasi tutto quello di cui avete bisogno per cominciare a scrivere le vostre avventure testuali è disponibile gratuitamente in rete. L'IF è oggi un hobby, non un lavoro; non dovete pensare di poter vendere i giochi che scriverete. Il pubblico delle avventure testuali è piuttosto piccolo: probabilmente solo alcune migliaia di persone in tutto il mondo sono avidi consumatori della IF contemporanea. Cercate il divertimento e la soddisfazione, non il profitto.

Anche se programmatori esperti possono apprezzare il considerevole impegno di creare un'avventura testuale dal nulla usando un linguaggio di programmazione generico come il BASIC o il C, molti sistemi specializzati per l'IF hanno già risolto buona parte dei problemi fondamentali relativi alla costruzione di un mondo. I sistemi più diffusi sono Inform di Graham Nelson (l'argomento di questo libro) e TADS (Text Adventure Development System) di Mike Roberts. Anche se ogni anno appaiono nuovi sistemi pieni di speranze, pochissimi raggiungono un'ampia diffusione; la maggior parte delle avventure testuali odierne (e praticamente tutte quelle generalmente considerate come ben scritte, mature, sofisticate, interessanti, innovative etc.) sono state create con uno di questi due sistemi. TADS è l'unica alternativa che regge il confronto con Inform quanto a popolarità, a capacità di gestire sia storie semplici che complesse, e a disponibilità su PC, Mac, palmari e molti altri calcolatori. Ma, visto che state leggendo queste pagine, assumeremo che abbiate già fatto una scelta e deciso di provare Inform.

Nella nostra breve guida, speriamo di dare le prime basi di Inform. Quando avrete appreso qualcosa sul sistema, sarete in grado di scrivere semplici avventure che potranno essere giocate dai vostri amici e, a mano a mano che diventerete più bravi, che potrete condividere (grazie alla Rete) con appassionati di tutto il mondo. Comunque, se vi interessa solo giocare[1] avventure scritte da

[1] Se vi sentite confusi a proposito dell'IF in generale o su questa distinzione tra scrivere e giocare, potreste dare un'occhiata al Capitolo 1 di questa guida "Che cos'è la narrativa interattiva" e

altri, piuttosto che scriverne voi, non c'è bisogno che impariate Inform e questa guida non vi serve.

Cosa vuole darvi questa guida

Poiché aspiriamo solo a darvi un'introduzione ad Inform e a che cosa esso può fare, tratteremo alcune sue caratteristiche solo superficialmente, mentre altre le ignoreremo del tutto. Il testo definitivo è l'*Inform Designer's Manual* (quarta edizione, maggio/luglio 2001) di Graham Nelson, noto comunemente come DM4; non potete sperare di usare con successo Inform senza avere al vostro fianco una copia di questo splendido libro. La nostra guida deve essere considerata solo come un supplemento al DM4: offre descrizioni passo passo di quegli aspetti di Inform che sono più importanti quando uno vi si avvicina. In ogni caso in cui sembra che discordiamo da quello che ha scritto Graham, sicuramente lui ha ragione e noi, beh, ci siamo confusi.

In quanto "tutorial", questa guida è pensata per essere stampata e letta sequenzialmente; non è pensata per essere usata online o come manuale di consultazione, anche se dà un compendio del linguaggio e delle librerie di Inform. Il nostro approccio consiste nell'insegnarvi qualcosa di Inform attraverso la creazione di tre avventure: tutte brevi, tutte giocabili completamente. "Heidi nella foresta" è più o meno l'avventura testuale più semplice possibile, ma riesce ugualmente a presentare vari concetti importanti. "Guglielmo Tell", una rivisitazione della famosa leggenda, è quasi altrettanto breve ma spazia di più nell'uso delle possibilità di Inform. Infine "Capitan Fato" presenta un supereroe da fumetti che ha urgente bisogno di cambiarsi. Nella guida parliamo di meno della metà delle possibilità di Inform, ma speriamo di aver almeno menzionato la maggior parte delle cose che è importante conoscere quando si inizia a scrivere la prima avventura in questo linguaggio.

Un'ultima cosa: Inform è un sistema potente, che spesso offre diversi modi per affrontare un particolare problema. Abbiamo cercato di presentare le cose nella maniera più semplice e coerente possibile, ma non dovrete sorprendervi se scoprirete altri approcci, magari più rapidi, magari più efficienti di quelli mostrati qui.

---

all'Appendice A "Come giocare un'avventura", od anche alle faq del newsgroup italiano it.comp.giochi.avventure.testuali presenti su http://www.ifitalia.info o alle Ifaq , in inglese, all'indirizzo http://www.plover.net/~textfire/raiffaq/ifaq/

Presentazione e stile

La maggior parte del testo è composto in questo carattere, tranne quando usiamo parole che fanno parte del sistema Inform (come `print`, `Include`, `VerbLib`) o di uno dei nostri giochi (come `uccello`, `nido`, `cima_albero`). La prima comparsa di un termine incluso nel glossario è in **grassetto**. Usiamo il corsivo per i "segnaposto": per esempio, bisogna interpretare l'enunciato Inform:

```
print "string";
```

come "mostra al giocatore il carattere o stringa di caratteri rappresentati qui dal segnaposto *string*". Per esempio:

```
[TYPE]

        print "Ciao mondo!";
        print "Quel ramo del lago di Como, che volge a mezzogiorno, tra
            due catene non interrotte di monti, [...] Ma se invece
            fossimo riusciti ad annoiarvi credete che non s'@`e fatto
            apposta.";
```

Abbiamo posto il tag `[TYPE]` prima dei pezzi di codice del gioco che potete digitare come parte funzionante degli esempi. Tale tag non ha nulla a che fare con Inform, ma è solo una nostra indicazione per differenziare il codice che lo segue dai pezzi di codice che verranno inseriti con l'unico obiettivo di illustrare alcune particolari caratteristiche di Inform.

Risorse utili in Rete

Una delle cose che daremo per scontate, oltre al vostro ardente desiderio di imparare Inform e la vostra abilità di lavorare senza problemi con file e cartelle sul vostro computer, è che abbiate accesso ad Internet. Ciò è piuttosto importante, dato che quasi tutto quello di cui avete bisogno è disponibile solo per questa via. In particolare, troverete molto materiale utile a questi indirizzi:

- `http://www.inform-fiction.org`

  La home page di Inform, gestita da Graham Nelson e un piccolo gruppo di collaboratori. Cosa più importante, vi potete trovare l'*Inform Designer's Manual* in formato PDF cliccando su "Welcome" e poi su "Manuals".

- `http://www.lulu.com`

  Copie rilegate in brossura dell'*Inform Designer's Manual* e dell'*Inform Beginner's Guide* (questo manuale in Inglese) oltre che della presente Guida possono essere acquistate presso questo sito di editoria "stampa su domanda".

- `http://www.firthworks.com/roger/`

  Le pagine su Inform di Roger Firth, tra cui l'Informary (che novità ci sono su Inform?) e le pagine delle FAQ su Inform (Inform Frequently Asked Questions, FAQ).

- `http://www.plover.net/~textfire/raiffaq/`

  Una lista più generale di FAQ in inglese sulla programmazione di avventure, sia in Inform che negli altri principali sistemi.

- `http://www.ifarchive.org`

  L'IF Archive, da cui si può scaricare quasi tutto ciò che c'è di gratuito e di pubblico dominio. Per una mappa cliccabile delle parti dell'Archivio relative all'Inform, vedi `http://www.firthworks.com/roger/informfaq/hh.html`.

  NOTA: prima dell'agosto 2001, l'IF Archive era ubicato altrove, e precisamente all'indirizzo `ftp://ftp.gmd.de/if-archive/`, ed è tuttora possibile trovare riferimenti a quell'indirizzo. Non usate il vecchio indirizzo: è probabile che le informazioni ancora disponibili in quel sito siano poco aggiornate.

- `news:rec.arts.int-fiction`

  Il newsgroup Usenet internazionale per autori di narrativa interattiva, comunemente abbreviato in RAIF. Qui troverete discussioni tecniche, critiche e di programmazione sull'IF, nonché assistenza rapida, amichevole ed esperta sui vostri dubbi del tipo "come faccio a. . . " (ma prima guardate il manuale, per piacere).

- `news:rec.games.int-fiction`

  Il newsgroup complementare per giocatori di avventure testali, noto anche come RGIF.

- `news:it.comp.giochi.avventure.testuali`

  ICGAT è il newsgroup italiano dedicato agli autori ed ai giocatori di avventure testuali italiani.

- `http://www.ifitalia.info`

  Il portale italiano dedicato all'interactive fiction e alle avventure testuali, raccoglie praticamente tutta la produzione italiana, moltissimi articoli, soluzioni e tanto altro ancora...

- `ftp://www.ifitalia.info/ifitalia/`

  L'archivio italiano FTP per le avventure testuali, il suo compito è offrire spazio web a tutti gli autori che ne fanno richiesta, inoltre raccoglie come backup tutto ciò che ha relazione con le avventure testuali italiane.

- `http://www.inform-italia.org/`

  Il sito di Giovanni Riccardi dedicato ad Inform e ad INFIT, la sua traduzione delle librerie. Su di esse (versione 2.5) ci si baserà in questo manuale.

- `http://www.vincenzoscarpa.it/inform/manuale`

  Il sito di Vincenzo Scarpa sul quale troverete il suo manuale dedicato ad Inform.

- `http://www.paololucchesi.it/`

  Il sito di Paolo Lucchesi sul quale potrete trovare alcune interessanti estensioni alle librerie da lui scritte.

- `http://www.slade.altervista.org`

  Il sito di Alessandro Schillaci sul quale potrete trovare JIF, un editor in Java per Inform, e l'Inform Pack un pacchetto autoistallante che contiene tutto ciò di cui si ha bisogno per iniziare a programmare con Inform. Nota: si controlli che le versioni dei vari programmi siano aggiornate.

- `http://www.terradif.net/`

  Sito della fanzine italiana d'interactive fiction scritta con la collaborazione dei membri del NG italiano delle avventure testuali. Ha iniziato le sue pubblicazioni nell'ottobre 2003. In molti numeri contiene interessanti articoli sulla programmazione in Inform.

Ringraziamenti

Diventare sufficientemente esperti di Inform da poterne parlare ad altri non è qualcosa che possa essere fatto rapidamente o da soli. Per arrivare dove siamo oggi, siamo stati aiutati in molte occasioni e in molti modi dalle persone notoriamente disponibili e amichevoli, troppo numerose per poterle elencare per nome, che fanno di `rec.arts.int-fiction` una risorsa tanto preziosa per la IF. Siamo grati a tutti.

Per la stesura di questa guida, abbiamo ricevuto aiuto specifico da molte persone (alcune delle quali neppure conosciamo): Harry Firth e Jo Quinn sono gli autori della copertina, mentre Barney Firth, Megan Firth e Phil Graham ci hanno assistito a proposito degli ambienti PC e Mac. Graham Nelson ha gentilmente scritto la Prefazione e ci ha deliziato con lunghi e dettagliati elenchi di utili commenti e suggerimenti su due delle stesure del libro; apprezziamo moltissimo anche i commenti di altri dei primi lettori, tra cui Jane d'España, Christine Firth, Jim Fisher, Muffy St. Bernard, Gunther Schmidl ed Emily Short. Paul Johnson, principiante nell'uso di Inform, ha messo alla prova la validità della guida come "tutorial" e ci ha coscienziosamente rassicurato sul fatto che effettivamente funziona. Dennis G. Jerz ha abilmente rivisto il testo, apportando innumerevoli migliorie alla nostra prosa spesso contorta e incoerente. Col suo abituale talento, David Cornelson sta curando il passaggio della guida alla rispettabilità di una stampa professionale. Grazie: è impossibile esagerare il valore di questo sostegno e aiuto dati spontaneamente.

Diamo credito alla generosità di `http://briefcase.yahoo.com` per aver reso la condivisione internazionale di file una passeggiata.

Infine, naturalmente, abbiamo un enorme debito di gratitudine nei confronti di Graham Nelson per aver creato il tutto, dandoci così l'opportunità, prima indipendentemente e poi in felice collaborazione, di usare e infine di presentare il sistema Inform per lo sviluppo di avventure testuali.

Roger Firth

Reading, Inghilterra

Sonja Kesserich

Madrid, Spagna

Agosto 2002

Note alla traduzione Italiana

Il presente volume, traduzione dell'originale *Inform Beginner's Guide*, nasce dall'iniziativa e dalla collaborazione di alcuni affezionati utenti di icgat, il newsgroup italiano dedicato alle avventure testuali, con il fine di mettere a disposizione degli aspiranti autori italiani di narrativa interattiva un volume che li potesse più facilmente introdurre alla programmazione in Inform. Vogliamo in particolare ringraziare: Daniele A. Gewurz (introduzione e capitolo 2), Paolo Lucchesi (capitoli 5, 6, 7, 8, 9, 11 e 12), Marco Falcinelli (capitoli 3, 4, 13, 14 appendici B, C, D), Paolo Vece (capitoli 1, 15, 16, 17, appendici A, E, F, G) e Giulio Veneziani (capitolo 10)[2]. Gli esempi in Inform sono stati portati in italiano da Paolo Lucchesi e corretti alla terza edizione da Marco Falcinelli.

Il presente volume è stato curato, aggiornato, rivisto ed impaginato da Marco Falcinelli, il quale coglie l'occasione per ringraziare Paolo Vece e Vincenzo Scarpa per aver corretto le bozze definitive ed aver contribuito alla realizzazione di un prodotto più professionale.

[2] Per chi è interessato alla genesi di questo manuale: le traduzioni base sono state realizzate nel corso del 2003 per iniziativa di una ragazza conosciuta sul newsgroup col nickname *Emerald*, l'autorizzazione alla traduzione fu richiesta da Tommaso Caldarola e gentilmente concessa da Roger Firth, i testi sono stati raccolti e pubblicati da Giovanni Riccardi sul proprio sito (l'indirizzo lo trovate nell'elenco fornito alle pagine precedenti). Le traduzioni pur slegate, non revisionate e non del tutto adattate alle librerie italiane hanno fornito a lungo un supporto per gli appassionati ed un'indispensabile base di partenza per la presente edizione. A tutti loro, e ai tanti che sostennero l'iniziativa creando un clima estremamente *propositivo* va un sentito ringraziamento.

1. Che cos'è l'interactive fiction?

Prima di cominciare a studiare Inform è forse meglio soffermarsi un po' su come l'IF, che ha molti elementi narrativi, sia differente dalla narrazione normale. Prima di farlo, però, vediamo l'esempio di una nota favola.

> "C'era una volta un uomo di nome Guglielmo Tell, veniva dalle Alpi Svizzere, vicino la città di Altdorf. Era sia un cacciatore che una guida, ed un provetto scalatore, viveva grazie alle sue abilità nel tiro con l'arco e nella mira. Capitò un giorno in cui Guglielmo andò in città per acquistare delle provviste, portando con sé il figlio Walter.
>
> A quel tempo la regione era governata da Hermann Gessler (un uomo vanitoso e mediocre, scelto per quella carica dall'imperatore austriaco), il quale, per dare sfoggio di potere nei confronti dei suoi assoggettati, aveva piazzato il suo cappello su un palo nella piazza della città, e tutti quelli che passavano erano costretti a rendergli onore. I cittadini riluttanti venivano "incoraggiati" da una truppa dei soldati del governatore, che si assicurava che gli inchini fossero sufficientemente rispettosi.
>
> Guglielmo sapeva del cappello, e dell'umiliante esercizio di obbedienza. Aveva sempre fatto in modo di evitare la piazza della città, sicuro del fatto che, vista la sua aperta ostilità per il governatore, il suo rifiuto ad inginocchiarsi gli avrebbe causato dei guai. Oggi, comunque, era costretto a passare vicino al cappello per raggiungere la conceria di Johansson.
>
> Non sapremo mai se Guglielmo sperava in un colpo di fortuna. La piazza era piena delle folle del giorno di mercato; i soldati erano particolarmente intenti nel loro lavoro d'incoraggiamento al rispetto, provocando chiunque con forti grida e qualche apprezzamento volgare. Guglielmo cinse la spalla del figlio con il braccio e si avviò in maniera decisa, ignorando tanto il palo che le guardie.
>
> Un soldato lo richiamò, ma Guglielmo fece finta di niente. Le altre guardie indirizzarono la propria attenzione sull'arciere. "Rendi omaggio al cappello", gli venne detto. Seguì un intenso silenzio. Guglielmo cercò di proseguire, ma ormai era circondato. Gli uomini lo conoscevano: uno gli consigliò di fare un discreto cenno del capo verso il cappello, e sarebbe finita lì. Tutti quelli che erano nelle vicinanze stavano guardando, quindi la mancanza di rispetto non poteva essere ignorata. Ci fu una lunga pausa. Guglielmo rifiutò.
>
> La notizia arrivò a Gessler, che si presentò in piazza con i rinforzi. Il piccolo uomo era estasiato all'idea di poter dare un esempio a questo cerca guai. Raccontò, deridendolo, delle molte abilità di Herr Tell, domandandosi se una tale bravura fosse la causa dell'orgoglio che gli impediva di riconoscere l'autorità dell'Imperatore. Il governatore comprendeva la cosa, e gli diede

una possibilità. Se Guglielmo fosse stato in grado di colpire una mela a cinquanta passi, Gessler sarebbe stato incline a mostrare pietà; d'altra parte, per rendere le cose interessanti, la mela doveva stare in equilibrio sulla testa di Walter.

Ogni cosa venne preparata. Guglielmo scelse una freccia, la sistemò e, lentamente, sollevò l'arco, cosciente dell'immobilità e del coraggio che Walter mostrava. Caricò, sentendo la tensione che partiva dalla corda e dalle dita, arrivava attraverso la mano fino al braccio. Aveva fatto dei tiri più difficili un passato, a cervi sfuggenti, o ad uccelli che spiccavano il volo; ma qui c'era di mezzo la vita di suo figlio... Non poteva sbagliare, non avrebbe sbagliato.

Guglielmo lanciò. La freccia volò dritta e sicura, fissando violentemente la mela all'albero che si trovava dietro il ragazzo. La folla esplose in un grido di sollievo ed ammirazione, mentre Gessler, contrariato, non ebbe altra possibilità che lasciarlo andare.

Anni dopo, Guglielmo guidò una rivolta contro il governatore... ma questa è un'altra storia."

E adesso un estratto della stessa storia, solo che questa volta è nella forma di una piccola avventura testuale. Se non sapete come s'interagisce con un'avventura testuale trovate qualche informazione generale nella sezione "Come si gioca con un'IF", e potete leggere una trascrizione completa del gioco su "Guglielmo Tell" nell'Appendice C:

```
Una strada di Altdorf
La piccola strada conduce verso nord alla piazza principale. La
gente del luogo sta sciamando nella città attraverso la porta a
sud, salutando a voce alta, offrendo prodotti in vendita,
scambiando notizie, informandosi con incredulità esagerata sui
prezzi delle merci esposte dai mercanti, i cui banchi rendono
ancora più difficile l'avanzare in mezzo alla folla.

"Stammi vicino, figliolo," dici, "altrimenti potresti perderti
fra tutta questa gente."

> VAI A NORD

Lungo la strada
La gente continua a premere e a farsi strada dalla porta sud
alla piazza principale, che si trova appena più a nord. Riconosci
la proprietaria di un banco di frutta e verdura.


Helga smette di sistemare le patate e ti saluta calorosamente.

"Salve, William, è una buona giornata per il commercio! Questo
è il giovane Walter? Come è cresciuto... Ecco, questa è una
mela per lui... Se toglie la parte ammaccata, il resto è buono.
Come sta la signora Tell? Salutamela davvero..."

> INVENTARIO
Stai portando:
    un mela
```

```
    un faretra (indossata)
    tre frecce
    un arco

> PARLA A HELGA
Ringrazi calorosamente Helga per la mela.

> DAI LA MELA A WALTER
"Grazie, Papi."

> NORD

Lato sud della piazza
La piccola strada che conduce verso sud si apre nella piazza
principale, per poi riprendere dalla parte opposta di questo
affollato luogo di ritrovo. Per continuare lungo la strada,
verso la tua destinazione - la conceria di Johansson - devi
attraversare, andando verso nord, la piazza, al centro della
quale vedi il cappello di Gessler messo su di un palo. Se vuoi
andare avanti, non puoi evitare di passarci vicino. Soldati
imperiali si fanno largo rudemente tra la folla, spingendo,
calciando e imprecando ad alta voce.
...
```

Alcune delle differenze più ovvie, sono individuate da queste domande:

- **Chi è il protagonista?**

  Il nostro esempio di prosa narrativa è scritto in terza persona; si riferisce all'eroe come "Guglielmo" e "lui", guardando e riportando le sue attività con distacco. In questo gioco IF d'esempio, impersonerete l'eroe, e vedrete tutto attraverso gli occhi di Guglielmo.

- **Che succede adesso?**

  La narrativa convenzionale è pensata per essere letta una volta, dall'inizio alla fine. A meno che non vi abbiate prestato attenzione la prima volta, generalmente non è necessario tornare indietro e leggere una frase una seconda volta; se lo fate, trovate esattamente lo stesso testo. L'autore vi guida attraverso il percorso che ha segnato; voi, i lettori, semplicemente seguite.

  Nell'IF, di solito, questo è molto meno vero. L'autore ha creato un paesaggio e lo ha riempito di personaggi, ma siete voi a scegliere quando e come esplorarlo. Il gioco avanza, almeno in maniera superficiale, sotto il vostro controllo; potreste esplorare prima la strada e poi la piazza, od entrare da un'altra parte. Di solito ci sono diversi percorsi da scoprire e seguire - e potete essere certi che non riuscirete a scoprirli tutti, almeno la prima volta.

- **Come funziona la cosa?**

  Sapete quando arrivate alla fine di un'opera di narrativa convenzionale: leggete l'ultima frase, e sapete che non ce ne saranno altre. Nell'IF è abbastanza chiaro quando raggiungente una fine; quello che è meno chiaro è

se quella sia l'unica conclusione possibile. Nella trascrizione del gioco d'esempio vincete colpendo la mela sulla testa di Walter. Ma se la doveste mancare? Che succederebbe se, per errore, lo colpireste? O se, invece, lanciaste all'odiato governatore? O se lasciaste la storia all'inizio, chinandovi di fronte al cappello del governatore e proseguendo per la vostra strada? Il gioco potrebbe terminare in uno qualunque di questi modi. La frase "che succede se" è la chiave per scrivere con successo, e dovrebbe sempre trovarsi in cima ai pensieri di uno scrittore d'IF.

- **Da dove viene fuori Helga?**

  Avrete notato che Helga ed il suo banco non compaiono nel normale flusso narrativo: è una diversione dal tema della storia. Ma nel gioco, lei assolve una serie di funzioni utili: menzionando i nomi "Guglielmo", "Walter" e "Frau Tell" (così che sappiate di chi parla la storia), presentando l'importantissima mela in un modo naturale e, soprattutto, fornendo un'opportunità per la "I" d'IF: una certa interattività. Senza quella (la possibilità d'interagire con l'ambiente della storia) il gioco è poco differente da un pezzo di narrativa convenzionale.

- **Quell'oggetto sembra interessante; mi puoi dire qualcosa di più?**

  Nella narrativa normale, quello che vedete è quello che avete; se voleste saperne di più in merito alla vita alpina nel quattordicesimo secolo, dovreste consultare un'altra fonte. L'IF, d'altra parte, offre almeno la possibilità di andare più a fondo, di osservare in dettaglio un oggetto che è stato solo menzionato. Ad esempio, avreste potuto dare un'occhiata al banco di Helga:

```
...Come sta la signora Tell? Salutamela davvero."

> ESAMINA IL BANCO
Davvero un piccolo banco, con un grosso mucchio di patate,
qualche carota, qualche rapa, un po' di mele.

> ESAMINA LE CAROTE
Prodotti locali.
```

  Vedete quelle descrizioni solo se le cercate; niente di quello che trovate è inaspettato, e se non esaminate il banco, non vi perdete niente di importante. Nondimeno avete aumentato l'illusione che stiate visitando un luogo reale. Questi dettagli diventerebbero tediosi se il banco e tutto quello che c'è sopra venisse descritto ogni volta che passate.

- **Come faccio a far funzionare questa cosa?**

  Sebbene la presenza di Helga sia una rielaborazione della storia originale, il lancio della freccia illustra il principio opposto: semplificare. La storia fa crescere la tensione descrivendo ogni passo che segue Guglielmo per prepararsi a lanciare. Va benissimo; è stato un arciere da tutta la vita, e sa come si fa. Voi, d'altra parte, ne sapete probabilmente poco o nulla di tiro con l'arco, e non ci si aspetta che tiriate ad indovinare con il funzionamento od il corretto vocabolario. Speriamo solo che sappiate che dovete lanciare

alla mela, ed è tutto quello che serve. Il gioco spiega che cosa succede, ma non vi costringe a leggere ogni singolo passo.

Naturalmente queste sono generalizzazioni, non verità universali; potete trovare degli ottimi lavori di IF che contraddicono tutte le osservazioni fatte. Comunque, per i nostri scopi di principianti nel mestiere di scrittori di IF, sono delle ottime differenze tra narrativa interattiva e quella convenzionale.

Torneremo sul "Guglielmo Tell" in un successivo capitolo, ma prima approfondiremo un esempio ancora più semplice. E prima di tutti e due, abbiamo bisogno di scaricare i file necessari che ci consentano di scrivere giochi in Inform.

2. I ferri del mestiere

La narrativa convenzionale, statica, può essere scritta non usando altro che carta e penna, o una macchina per scrivere, o un elaboratore di testi; per produrre Narrativa Interattiva invece serve qualcosa di più, e il processo creativo è lievemente più complesso.

- Nella narrativa statica, prima si scrive il testo, e poi lo si controlla leggendo ciò che si è scritto.
- Nell'IF, si deve comunque scrivere tutto il testo, ma bisogna anche stabilire quale testo viene mostrato ed in quale momento. Una volta che si sono scritte le istruzioni necessarie in Inform, si usa un programma compilatore per convertirle in un formato giocabile. I dati risultanti vengono giocati mediante un programma interprete, che permette di interagire con il mondo che è stato sviluppato.

Nella narrativa classica Ciò Che Scrivi È Ciò Che Leggi (come nei programmi di tipo WYSIWYG, "What You See Is What You Get"), mentre nell'IF il formato in cui si scrive inizialmente il gioco non somiglia molto al testo che l'interprete mostra alla fine. Il gioco "Guglielmo Tell", per esempio, nella forma in cui lo abbiamo scritto, comincia così:

```
!=============================================================
Constant Story "Guglielmo Tell";
Constant Headline
            "^Semplice esempio in Inform
             ^di Roger Firth and Sonja Kesserich.^";
             ! Traduzione di Paolo Lucchesi

Release 1;  Serial "020428";
! per tener conto delle release pubbliche

Constant MAX_SCORE = 4;
Include "Parser";
Include "VerbLib";
Include "Replace";
!=============================================================
! Classi

Class   Room
  has   light;
...
```

Non ci sarà mai bisogno di vederlo nella forma prodotta dal compilatore:

```
050000012C6C2C2D1EF6010A0C4416900010303230313031004253FEA90C0000000
00000000000000000000000000168F0000000000000102000000000362E3231

...
```

ma, come si vedrà dalla trascrizione integrale di "Guglielmo Tell" nell'Appendice C, il giocatore vedrà quello che segue:

```
Il luogo: Altdorf, nel cantone Svizzero di Uri. Siamo nell'anno
1307, e la Svizzera è sotto il governo dell'imperatore Alberto di
Asburgo. Il governatore locale - il balivo - è il gradasso Hermann
Gessler, che ha posto il suo cappello su di un palo di legno nel
centro della piazza principale; chiunque passi attraverso la piazza
deve inchinarsi a questo odioso simbolo del potere imperiale.

...
```

Ovviamente, per scrivere IF bisogna fare molto più che scrivere le parole una dopo l'altra. Per fortuna, possiamo immediatamente semplificare le cose: è previsto che nella forma tradotta (quella prodotta dal compilatore Inform), i numeri e lettere criptiche memorizzate in quello che si chiama **story file**, siano letti solo dall'interprete. Lo story file è un esempio di file "binario", che contiene i dati che verranno usati solo da un programma. Ci si può scordare di tutta quella roba illeggibile.

Così rimane solo la prima forma, quella che inizia con "Constant Story", che rappresenta la storia scritta come testo di un'IF. Questo è il **file sorgente** (così detto perché contiene il gioco nella sua forma originale, vista alla sorgente) che creerete nel vostro computer. Il file sorgente è un file di testo (o "ASCII"), contenente parole e frasi che possono essere lette da esseri umani, anche se con un minimo di preparazione, che è esattamente ciò che vuol darvi questa guida.

Come si crea questo file sorgente? Usando un terzo programma: un editor di testi. Però, a differenza del compilatore e dell'interprete, questo programma non è specifico per il sistema Inform, e neppure per l'IF. Un editor è uno strumento completamente generale per creare e modificare file di testo; probabilmente ne avete già uno (seppur molto semplice) sul vostro computer (in un PC con Windows c'è il "Blocco Note", mentre un Macintosh ha "SimpleText" o "TextEdit"), e se ne può scaricare uno migliore da Internet. Un editor è come un elaboratore di testi (come "MS Word"), ma molto meno complesso; non prevede elaborate funzioni d'impaginazione, né il controllo sul grassetto, il corsivo o i caratteri usati, né immagini da inserire; dà solo la possibilità di scrivere righe di testo, il che è esattamente ciò che serve per creare un gioco d'IF.

Se si guarda il sorgente del gioco nella pagina precedente, o il listato di "Guglielmo Tell" nell'Appendice C, si noterà `Include "Parser";` e `Include "VerbLib";` poche righe dopo l'inizio del file. Queste sono le istruzioni per dire al compilatore Inform di "includere", cioè inserire il contenuto dei file chiamati `Parser.h` e `VerbLib.h`. Questi non sono file che dovete creare; sono **file di libreria** (o librerie) standard, che fanno parte del sistema Inform. Tutto quello che si deve fare è ricordarsi di includerli in ogni avventura che si scrive. Finché non è abbastanza chiaro come funziona Inform, non c'è bisogno di preoccuparsi di che cosa contengono (ma è possibile guardarli se si vuole: sono file di testo leggibili, come quelli che questa guida vi insegnerà a scrivere).

NOTA: `Replace.h` è un file specifico della libreria Italiana `INFIT` (che sta per INForm in ITaliano) ovvero la libreria su cui si baserà questo volume per

l'adattamento alla lingua italiana. Bisogna ricordarsi di includerlo sempre dopo `Parser` e `Verblib` perché contiene delle funzioni contenute in questi file, ma tradotte e adattate nella nostra lingua.

Abbiamo quindi presentato tutti i pezzi di cui c'è bisogno per scrivere un'avventura in Inform:

- un **editor** di testi che possa creare e modificare il **file sorgente** che contiene le descrizioni e le definizioni del vostro gioco. Anche se non è consigliabile, si potrebbe persino usare un elaboratore di testi per far ciò, ma bisogna ricordarsi di salvare il gioco nel formato "solo testo";
- alcuni **file di libreria** Inform che includerete nel file sorgente del vostro gioco per definire il **world model**, un ambiente di gioco essenziale e molte ed utili definizioni standard. Questi file includono quelli della libreria standard e quelli di INFIT, necessari per scrivere le avventure in italiano;
- il **compilatore** Inform, che legge il vostro file sorgente (e i file di libreria) e traduce le vostre descrizioni e definizioni in un altro formato, lo **story file**, che sarà letto solo da...
- un **interprete** Inform, che è ciò che useranno i giocatori della vostra avventura. Un giocatore non ha bisogno dei file sorgente, di libreria o del compilatore, ma solo dell'interprete e del gioco in formato compilato (che, essendo in un formato binario privo di senso per un essere umano, ha il pregio di impedire ai giocatori di barare).

Tutte queste cose, tranne l'editor, possono essere scaricate gratuitamente dall'IF Archive[3]. Una possibilità è scaricarle una per una, seguendo le indicazioni della pagina di G. Nelson: visitate `http://www.inform-fiction.org/inform6.html` cliccate su "Welcome" e poi "Software". Però, se usate un PC, troverete più semplice scaricare un pacchetto completo che contiene tutto quello che vi serve per cominciare.

Inform su un PC IBM

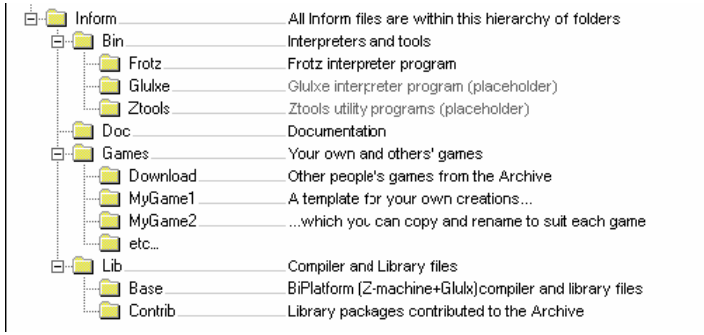
Seguite queste istruzioni:

1. Scaricate il GIP Inform Pack per PC da IF Italia in una directory provvisoria del vostro PC.
2. Usate un programma come WinZip[4] per scompattare il file scaricato, creando così una nuova cartella Inform. Spostate questa cartella (e il suo contenuto) in una directory appropriata: un buon posto potrebbe essere

[3] Una buona soluzione è anche quella di scaricare l'Inform Pack dal sito di Alessandro Schillaci o da quello di IF Italia.

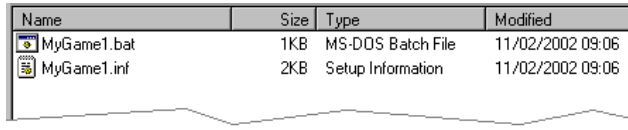
[4] Consigliamo ZipGenius un programma gratuito molto efficiente ed analogo a Winzip.

C:\Documenti\Inform, ma potreste anche usare C:\Inform o C:\Programmi\Inform. Ora dovreste avere questo insieme di cartelle:



Per rendere lo scaricamento snello e veloce, queste cartelle comprendono lo stretto indispensabile per cominciare come autore Inform: il compilatore e l'interprete, i file di libreria, il file di esempio `Ruins.inf` presentato nell'*Inform Designer's Manual* e un modello da seguire per il proprio primo gioco[5]. Ci sono anche delle altre cartelle messe come segnaposto in cui scaricare componenti addizionali, se e quando li vorrete. Prima possibile dovreste scaricare l'*Inform Designer's Manual* nella cartella `Inform\Doc`: è un testo essenziale da possedere, e non è stato incluso in questo pacchetto solo per le sue dimensioni (3,2MB).

3. Per verificare che i file scaricati funzionino correttamente, usate l'Esplora Risorse di Windows per vedere il contenuto della cartella `Inform\Games\MyGame1`: vedrete i due file `MyGame1.bat` e `MyGame1.inf`:



`MyGame1.inf` è una minuscola avventura ridotta all'osso in formato sorgente per Inform. Per convenzione, tutti i file sorgente in Inform hanno come estensione `.inf`; Windows ha una sua definizione per i file `.inf`, e perciò li associa al tipo "Setup Information", ma sembra che questo non crei problemi. Se cliccate due volte sul file, dovrebbe aprirsi nel "Blocco Note" e così potete vedere com'è fatto, anche se probabilmente per ora non vi dirà molto. Noterete nella cartella anche i file `MyGame1_it.inf` e `MyGame1_it.bat`: essi sono analoghi ai primi due, ma con la differenza di essere stati adattati e tradotti in italiano.

[5] E la sua traduzione in italiano `Ruins_it.inf`, curata da Vincenzo Scarpa e Raffaello Valesio.

4. `MyGame1.bat` è un file di "batch" MS-DOS (un vecchio tipo di programma di solo testo, risalente ai giorni precedenti le interfacce grafiche) che fa partire il compilatore Inform. Cliccatelo due volte; si apre una finestra DOS mentre il gioco viene compilato e vedrete:

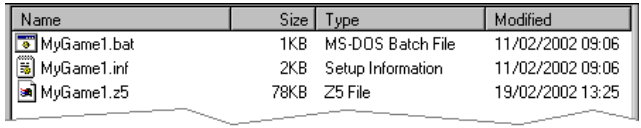
```
C:\Documenti\Inform\Games\MyGame1>..\..\Lib\Zcode\Inform
MyGame1  +include_path=.\,..\..\Lib\Zcode,..\..\Lib\Contrib
| more

PC/Win32 Inform 6.21 (30th April 1999)

C:\Documenti\Inform\Games\MyGame1>pause  "alla  fine  della
compilazione"

Press any key to continue . . .
```

   Premete lo spazio e chiudete la finestra DOS.

   `MyGame1_it.bat` è identico al precedente con l'unica differenza che come file di gioco compila `MyGame1_it.inf` e che aggiunge l'opzione di compilazione `+language_name=italian` che dice al compilatore, appunto, che il linguaggio usato nel gioco è l'italiano.

   NOTA: Sotto Windows NT, 2000 e XP, la finestra DOS si chiude automaticamente quando premete lo spazio.

5. È comparso nella cartella uno "story file": `MyGame1.z5` (e `MyGame1_it.z5` se avete compilato anche la versione italiana); si tratta del gioco compilato, che potrete eseguire con un interprete:



   L'estensione `.z5` significa che il file contiene un gioco per la Z-machine nel formato per la versione 5 (lo standard attuale).

6. Usate l'Esplora Risorse di Windows per visualizzare il contenuto della cartella `Inform\Bin\Frotz` e cliccate due volte su `Frotz.exe`; l'interprete aprirà una finestra vuota.

7. Selezionate `File > Open New Story...`, sfogliate fino a mostrare la cartella `Inform\Games\MyGame1` e selezionate il file `MyGame1.z5` (o `MyGame1_it.z5`). Cliccate `Open`. Il gioco partirà nella finestra di WinFrotz.

8. Quando vi sarete stancati di "giocare" all'avventura d'esempio, il che succederà presto, potrete digitare il comando ESCI, selezionare `File > Exit`, o semplicemente chiudere la finestra di Frotz.

9. Usando le stesse tecniche potete compilare e giocare `Ruins.inf` (o la sua traduzione in italiano), che si trova nella cartella `Inform\Games\Download`. RUINS è il gioco usato come esempio per tutto l'*Inform Designer's Manual*[6].

Definire a che programmi sono associati i file

Aprire l'interprete e poi trovare il file dell'avventura che si vuole giocare è una procedura inelegante e scomoda da compiere ogni volta. Probabilmente troverete più semplice associare automaticamente i file la cui estensione è .z5 all'interprete WinFrotz.

1. Cliccate due volte su `MyGame1.z5`; Windows vi chiederà con quale programma lo dovrà aprire:
   - scrivete `'Gioco per la Z-machine'` come "`Descrizione...`"
   - cliccate su "`Usa sempre questo programma...`"
   - cliccate "`Altro...`"
2. Sfogliate fino a trovare la cartella `Inform\Bin\Frotz` e selezionate `Frotz.exe`. Cliccate Apri. D'ora in poi potrete giocare un'avventura semplicemente cliccando due volte sul suo file `.z5`.

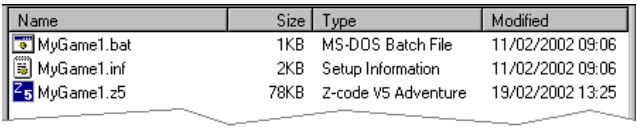
Cambiare l'icona

Se l'icona che Windows mostra per `MyGame1.z5` non sembra avere l'aspetto giusto, la potete cambiare.

1. In Esplora Risorse, scegliete `Strumenti > Opzioni Cartella...` e cliccate `Tipi di file`:
   - scegliete il tipo di file relativo al gioco, nella lista che è in ordine o di applicazione (FROTZ) o di estensione (Z5)
   - cliccate `Avanzate`...
2. Nella finestra di dialogo `Modifica Tipo di File` cliccate `Cambia Icona`.
3. Nella finestra di dialogo `Cambia Icona`, assicuratevi che il nome del file sia `Inform\Bin\Frotz\Frotz.exe`, e selezionate una delle icone mostrate. Cliccate OK per chiudere le finestre di dialogo.

I file nella cartella ora dovrebbero avere questo aspetto:

[6] La traduzione in italiano di Ruins ad opera di Vincenzo Scarpa costituisce invece oggetto di analisi di alcuni capitoli del suo manuale su Inform.



Name	Size	Type	Modified
MyGame1.bat	1KB	MS-DOS Batch File	11/02/2002 09:06
MyGame1.inf	2KB	Setup Information	11/02/2002 09:06
MyGame1.z5	78KB	Z-code V5 Adventure	19/02/2002 13:25

Compilare usando un file batch

Potete vedere, e naturalmente cambiare, il contenuto di `MyGame1.bat`, il file di batch che cliccate due volte per far partire il compilatore, usando qualunque editor di testi. Vedrete due righe, più o meno così (la prima parte è tutta su un'unica lunga riga, con uno spazio tra `-S` e `+include_path`):

```
..\..\Lib\Zcode\Inform MyGame1
          +include_path=.\,..\..\Lib\Zcode,..\..\Lib\Contrib | more
pause "alla fine della compilazione"
```

NOTA: nella versione italiana `MyGame1_it.bat` subito prima del nome del file `MyGame1_it` compare un'ulteriore istruzione, ovvero `+language_name=italian` : essa dice al compilatore che ci si accinge a compilare un file (`MyGame1_it.inf`) che richiama le librerie di traduzione italiane.

Queste lunghe stringhe di testo sono righe di comando, una potente interfaccia che risale a prima delle icone e dei menu noti alla maggior parte degli utilizzatori di computer. Non ci sarà bisogno di padroneggiare l'interfaccia a riga di comando per iniziare ad usare Inform, ma questo paragrafo vi spiegherà che cosa fanno queste particolari righe di comando. La prima riga ha quattro parti (cinque nella versione italiana):

1. `Inform` si riferisce al compilatore e `..\..\Lib\Zcode` è il nome della directory che lo contiene (con l'indirizzo scritto relativamente a questa directory, quella che contiene il file sorgente).
2. Nel caso si stia per compilare un file di codice in italiano si aggiunge l'opzione `+language_name=italian`.
3. `MyGame1` (o `MyGame1_it`) è il nome del file sorgente in Inform; non è indispensabile specificarne l'estensione .inf.
4. `+include_path=.\,..\..\Lib\Zcode,..\..\Lib\Contrib` dice al compilatore dove cercare i file da includere come `Parser` e `VerbLib`. Vengono suggeriti tre posti: la directory attuale, che contiene il file sorgente (`.\`); la directory che contiene i file standard di libreria (`..\..\Lib\Zcode`); la directory che contiene librerie aggiuntive utili dovute al contributo della comunità Inform. La ricerca avviene in queste tre directory in quest'ordine.

   Nota: sulla linea di comando talvolta può capitare di trovare degli **switch** come ad esempio `-S`, usati per controllare particolari aspetti della compilazione. Invece di porre tali opzioni di compilazione qui, troviamo che sia molto più semplice e conveniente aggiungere gli switch che

riteniamo necessari subito all'inizio del file sorgente, come spiegheremo nel prossimo paragrafo.

5. `| more` fa sì che il compilatore faccia una pausa se trova più errori di quelli che può riferire in una singola schermata, anziché farli scorrere via dalla finestra MS-DOS. Bisogna premere lo spazio per far proseguire la compilazione.

La seconda riga, `pause "alla fine della compilazione"`, serve solo per impedire che la finestra si chiuda prima che se ne possa leggere il contenuto, come altrimenti farebbe in Windows NT, 2000 e XP.

Dovrete scrivere un nuovo file batch come questo per ogni nuovo file sorgente che creerete. L'unica parte che cambierà nel nuovo file è il nome del file sorgente Inform, `MyGame1` in questo esempio, e l'inserimento o meno della specifica della lingua a seconda che vi accingiate a scrivere un gioco in italiano o in inglese. Il nome del gioco va cambiato e deve corrispondere al nome del nuovo file sorgente; tutto il resto può rimanere uguale in ogni file `.bat` che create.

Procurarsi un editor migliore

Anche se NotePad va bene per cominciare, la vita sarà molto più semplice se vi procurerete un editor più potente. Raccomandiamo TextPad, disponibile come shareware su `http://www.textpad.com/`; in più, alla pagina `http://www.onyxring.com/informguide.aspx?article=14` ci sono le istruzioni dettagliate su come migliorare il modo in cui TextPad lavora con Inform.

La maggior miglioria, quella che rende lo sviluppo di avventure enormemente più semplice, consiste nel poter compilare il file sorgente dall'interno dell'editor. Non c'è bisogno di salvare il file, passare ad un'altra finestra e cliccare due volte sul file batch (e anzi, non serve proprio più il file batch): basta premere un tasto mentre si lavora sul file, e viene compilato lì per lì.

Si può anche far girare l'interprete con la stessa facilità. La comodità di far ciò controbilancia di gran lunga il breve tempo necessario per scaricare e configurare TextPad.

Oltre al TextPad potete tenere presente anche il Crimson Editor, che è open source, che ha le medesime funzionalità o ancora meglio JIF. Quest'ultimo è un editor espressamente progettato da Alessandro Schillaci per elaborare sorgenti di Inform, compilarli e lanciare l'interprete il tutto all'interno dell'editor. In più, essendo scritto in Java può essere istallato su praticamente ogni piattaforma o sistema operativo possediate. Per la sua istallazione rinviamo alle istruzioni presenti sul sito di JIF `http://www.slade.altervista.org/JIF`.

Oltre a quelli che raccomandiamo, altri buoni editor vengono elencati nella pagina di FAQ a `http://www.plover.net/~textfire/raiffaq/`. Una funzione che è utile tenere d'occhio è il "syntax colouring", grazie al quale l'editor capisce

le regole sintattiche di Inform per contraddistinguere, con colori diversi, alcuni elementi del vostro file sorgente, come ad esempio il rosso per le parentesi quadre, graffe e tonde `[ ]` `{}` e `( )`, blu per le parole riservate come `Object` e `print`, verde per le parti tra virgolette come `'...'` e `"..."` e così via. La "colorazione sintattica" è di grande aiuto per scrivere un file sorgente corretto ed evitare stupidi errori di compilazione.

Altre cose da sapere sui compilatori

Il compilatore Inform è un programma potente ma non onnipotente; fa molto lavoro, ma lo fa tutto insieme, senza fermarsi a fare domande. Il suo input è un file sorgente in forma di testo leggibile; l'output è un file binario noto come file in **Z-code** (perché contiene il gioco tradotto in codice per la Z-machine, che descriviamo nella prossima sezione).

Se si è fortunati, il compilatore tradurrà il vostro file sorgente in Z-code; ci si può forse meravigliare che non mostri alcun messaggio di successo quando ci riesce. D'altronde, spesso fallisce, a causa di errori fatti scrivendo l'avventura. Inform definisce un insieme di regole (una lettera maiuscola qui, una virgola lì, queste parole solo in un certo ordine, quelle parole scritte esattamente nel tal modo) sulle quali il compilatore è estremamente pignolo. Se si infrangono queste regole, il compilatore si lamenta: si rifiuta di scrivere un file in Z-code.

Non vi preoccupate: è facile imparare le regole, ma è altrettanto facile violarle e tutti i programmatori in Inform senza volere lo fanno regolarmente. Ci sono ulteriori informazioni su come affrontare questi errori e come controllare il comportamento del compilatore nel Capitolo 15 "Compilare la vostra avventura".

Altre cose da sapere sugli interpreti

Uno dei grandi vantaggi del funzionamento di Inform è che un'avventura compilata, il file in Z-code, è "portabile" tra computer diversi. Non solo da un PC ad un altro: esattamente lo stesso file girerà su un PC, un Mac, un Amiga, macchine UNIX, mainframe IBM, palmari con Palm OS e su dozzine di altri calcolatori passati, presenti e futuri. La magia che rende questo possibile è nell'interprete, un programma che finge di essere un semplice computer chiamato **Z-machine**. La Z-machine è un calcolatore immaginario (o "virtuale"), ma la sua struttura è stata specificata con grande cura cosicché un programmatore esperto ne può costruire uno con relativa semplicità. Ed è esattamente ciò che è accaduto: un guru del Macintosh ha creato un interprete Inform che gira sui Mac, un mago dello UNIX ne ha creato uno per le workstation UNIX e così via. Qualche volta si ha addirittura la scelta; per macchine diffuse come i PC e i Mac sono disponibili vari interpreti. E la cosa meravigliosa è che ognuno di questi interpreti, su ognuno di questi calcolatori, è

in grado di giocare ogni gioco che è mai stato scritto in Inform e anche, come sorpresa aggiuntiva, anche tutti i classici giochi Infocom degli anni '80 come "Zork" e "The Hitchhiker's Guide to the Galaxy".

Un nuovo interprete che merita attenzione è Gargoyle, meglio se nella sua versione modificata (Gargoyle_mod) da Lorenzo Marcantonio, che ha una resa tipografica a schermo dei caratteri rendendo assai più piacevole la lettura dei testi. Potete scaricare tale interprete su `http://www.logossrl.com/gargoyle-mod/gargoyle_mod_it.html`.

(In realtà, l'ultima frase è una lieve esagerazione; alcuni giochi sono molto lunghi, o contengono delle immagini, e non tutti gli interpreti sono in grado di gestire ciò. Comunque, con questa piccola riserva, la cosa è vera.)

Basta chiacchiere: cominciamo! È ora d'iniziare a scrivere la nostra prima avventura.

3. Heidi, il nostro primo gioco in Inform

Ognuno dei tre giochi presenti in questa guida sarà esaminato passo dopo passo, al fine di facilitare l'apprendimento; per trarre il maggior beneficio dai nostri esempi consigliamo, soprattutto se si è agli inizi, di non limitarsi a leggere il libro, ma di esercitarsi scrivendo il codice sul proprio computer. Il primo gioco, descritto in questo capitolo e nei due che seguiranno, narra una breve e romantica storia:

> "Heidi vive in una piccola baita nel profondo della foresta. Un bel giorno, mentre riposa davanti alla sua baita, sente il frenetico cinguettio di un piccolo passerotto, il cui nido è caduto da un grande albero nella radura! Heidi mette l'implume nel nido, e quindi si arrampica sull'albero e adagia il nido sul suo ramo. Ahhh non è tenerissima?"

Sembrerebbe davvero semplice, ma anche così dovremo sudare parecchio prima di farne un gioco in Inform completo e soddisfacente. Proseguendo per gradi, abbozzeremo innanzi tutto la storia e quindi successivamente rifiniremo tutti i dettagli fino a quando non otterremo un risultato che soddisferà le nostre ambizioni, considerato che siamo al nostro primo lavoro in Inform (ci sarà tempo più tardi per aggiungere cose più interessanti).

Creare un file sorgente di base

Il primo obiettivo è creare un modello di file sorgente di Inform. Ogni gioco che progetteremo inizierà con i seguenti passaggi:

1. Create una cartella `Inform\Games\Heidi` (magari copiando la cartella `Inform\Games\MyGame1`).
2. Nella cartella, usate il vostro editor di testo per creare il file `Heidi.inf`:

```
[TYPE]
    !% -SD
    !=============================================================================
    Constant Story "Heidi";
    Constant Headline
                "^Semplice esempio in Inform
                 ^di Roger Firth e Sonja Kesserich.^";
                 ! Traduzione di Paolo Lucchesi
    Constant MAX_CARRIED 1;

    Include "Parser";
    Include "VerbLib";
    Include "Replace";

    !=============================================================
    ! Oggetti di gioco

    !=============================================================
    ! Entry point routines
```

```
[ Initialise; ];

!=============================================================
! Grammatica

Include "ItalianG";

!=============================================================
```

Presto spiegheremo quale è il significato di queste istruzioni. Per adesso limitiamoci a scrivere il codice nel file di testo `heidi.inf` facendo particolare attenzione agli otto punti e virgola ed assicurandoci di chiudere tutte le virgolette `"......"`. Le linee che iniziano con il punto esclamativo sono invece dei semplici commenti con il compito di rendere più semplice la lettura del codice.

Controllate che il nome del file sia `Heidi.inf`, e non `Heidi.txt` o `Heidi.inf.txt`.

Ricordate che nella redazione di questa guida è stato posto il tag `[TYPE]` prima di ogni pezzo di codice che noi raccomandiamo di scrivere nel file del vostro gioco mentre leggete questi esempi (il che significa che *non dovete* digitare il codice che non è preceduto da tale simbolo). Imparerete Inform più velocemente provando il vostro codice e non limitandovi a leggere pedissequamente questa guida.

3. Nella stessa cartella, usate il vostro editor di testo per creare il file di supporto per la compilazione del sorgente `Heidi.bat` (su un PC le istruzioni prima di "pause" sono un'unica lunga linea):

```
[TYPE]

..\..\Lib\Zcode\Inform +language_name=italian Heidi +
      include_path=.\,..\..\Lib\Zcode,..\..\Lib\Contrib | more

pause "alla fine della compilazione"
```

Ricordate che vi è un solo singolo spazio tra `+language_name` e `Heidi` e tra `Heidi` e `+include_path`.

Digitate il codice nel file o copiate e incollate dal file `MyGame1_it.bat`, facendo poi le opportune modifiche al nome del file. A questo punto, dovreste avere una cartella `Heidi` contenente due file: `Heidi.inf` e `Heidi.bat`.

4. Compilate il file sorgente `heidi.inf` (fate riferimento a "Inform su un PC IBM" al capitolo precedente.) Se la compilazione funziona, viene creato il file dell'avventura `heidi.z5` nella cartella. Se la compilazione *non funziona*, avete probabilmente commesso qualche errore di digitazione nel testo; controllate fino a trovarlo.

5. Adesso, lanciando il file dell'avventura nel vostro interprete per Inform, dovreste vedere esattamente le seguenti righe (ad eccezione del Serial number che sarà differente essendo basato sulla data):

```
Heidi
Un semplice esempio in Inform
di Roger Firth e Sonja Kesserich.
Release 1/ Serial number 050804/ Inform v6.31 Library 6/11 SD

Oscurità.
E' completamente buio, e non riesci a vedere niente.

>
```

Quando otterrete questo risultato, il vostro file sorgente sarà corretto. Andiamo ora a spiegarne il contenuto.

Comprendere il file sorgente

Sebbene ognuno di noi possegga un approccio personale e una propria libertà d'espressione, i file sorgenti tendono tutti a conformarsi allo standard, soprattutto per quanto riguarda la struttura: le prime linee d'inizio, il pezzo di codice alla fine, e tutta quella roba che c'è in mezzo. Ciò che intendiamo fare è una mappa della struttura del file sorgente in modo da avere un quadro chiaro nel quale disporre ogni elemento del gioco. Avere una buona visione di questa mappa fin dall'inizio ci aiuterà a tenere ben organizzato e comprensibile il codice a mano a mano che svilupperemo il gioco.

Possiamo individuare una mezza dozzina di regole di Inform solo guardando il nostro piccolo codice sorgente.

- Se la *primissima linea* (o linee) del file sorgente inizia con il simbolo "`!%`", allora il compilatore tratterà ciò che segue su tali linee come istruzioni di controllo per se stesso invece che come parte del sorgente del gioco. Le istruzioni più comunemente inserite in questo punto sono switch del compilatore, ovvero un modo per controllare gli aspetti particolari delle sue operazioni. I due switch che abbiamo inserito (ve ne sono molti a disposizione) nel codice precedente sono tra i più comuni; essi predispongono lo "**Strict mode**", che rende meno facile al gioco comportarsi male in fase di esecuzione, e la modalità **Debug**, che prevede alcuni comandi extra che possono rivelarsi utili quando si cerca di risolvere dei problemi.

  Nota: in realtà, `-s` è ridondante, dal momento che lo "Strict mode" è già attivo di default. Lo abbiamo incluso qui come promemoria per ricordarci che (a) per *disattivarlo* è necessario inserire lo switch `-~s`, (b) che c'è differenza tra la lettera maiuscola e minuscola; infatti, `-s` provoca la visualizzazione delle statistiche (mentre `-~s` non fa nulla).

- In ogni altro caso, quando il compilatore incontra un punto esclamativo, ignora tutto il contenuto che lo segue sulla riga. Se il `!` è all'inizio della linea,

l'intera riga sarà ignorata; se il `!` è a metà della linea il compilatore terrà conto della prima metà di essa ed ignorerà qualsiasi cosa vi sia dopo il punto interrogativo. Per questo il punto esclamativo viene utilizzato per commentare il codice; i **commenti** sono utili per ricordarci come funziona un certo passaggio o perché abbiamo affrontato un problema in una certa maniera. La lunga sfilza di segni di uguaglianza che abbiamo inserito nel codice, perciò, non hanno nulla di speciale, sono lì unicamente per formare una linea di demarcazione tra pezzi di codice.

È sempre una buona idea quella di commentare il codice: anche se avete tutto chiaro nella testa quando lo scrivete, considerate che tra qualche mese probabilmente vi sarete dimenticati tutto.

Naturalmente il compilatore *non* riserva nessun trattamento speciale ai punti interrogativi presenti nel testo quotato: un `!` tra virgolette `"..."` è trattato come un normale carattere. Su questa linea, ad esempio, il primo `!` è parte della sequenza (o **stringa**) di caratteri che viene visualizzata, mentre il secondo, fuori dal testo quotato, preannuncia un commento:

```
print "Ciao mondo!"; ! <- è invece per il commento
```

- Il compilatore ignora le righe bianche, e tratta gli spazi ripetuti come un singolo spazio (ad eccezione di quando gli spazi sono parte di una stringa). Queste due regole ci dicono che *potremmo* avere un file sorgente che assomiglia a questo:

```
Constant Story "Heidi";
Constant Headline
"^Un semplice esempio in Inform^di Roger Firth e Sonja
Kesserich.^";
Include "Parser";Include "VerbLib";Include "Replace";
[ Initialise; ];
Include "ItalianG";
```

  Noi abbiamo preferito non digitarlo in questa maniera perché, sebbene più corto, è molto più difficile da leggere. Quando si progetta un gioco, si spende un sacco di tempo a studiare il codice che si scrive, così è sempre meglio organizzarlo al meglio ed in modo che sia il più leggibile possibile.

- Ogni gioco ha bisogno della definizione di alcune **constanti**: `Story` (il nome del gioco) e `Headline` (normalmente informazioni sul tema dell'avventura, il copyright, i nomi degli autori e così via). Questi due valori **string**, insieme al numero della versione, alla data e ai dettagli del compilatore, compongono il **banner** che viene visualizzato all'inizio di ogni partita.

- Ogni gioco ha bisogno di quattro linee che iniziano con il comando `Include` che includono nel file sorgente le librerie standard:

```
Include "Parser";
Include "VerbLib";
Include "Replace";
...
Include "ItalianG";
```

Devono sempre essere disposte in questo ordine, con `Parser` e `VerbLib`, quest'ultima seguita da `Replace`, vicino all'inizio del file, e `ItalianG` vicino alla fine.

NOTA: Tale schema è proprio dei giochi compilati per l'italiano usando le librerie INFIT che sono state incluse del pacchetto menzionato al capitolo precedente. I sorgenti inglesi raramente prevedono il file `Replace.h` ed al posto del file `ItalianG.h` hanno il file `Grammar.h`.

- Ogni gioco ha bisogno di definire una routine `Initialise` :

```
[ Initialise; ];
```

La **routine** che abbiamo definito non fa nulla di utile, ma in ogni caso è necessario che sia presente. Più tardi, torneremo su `Initialise` e spiegheremo che tipo di routine è, e perché ne abbiamo bisogno.

- Avrete notato che alla fine di ogni istruzione menzionata nelle precedenti tre regole vi è un punto e virgola. Inform è veramente pignolo nella sua punteggiatura e non sarà affatto contento se vi dimenticate un punto e virgola alla fine di un'istruzione. Infatti, il compilatore continua a leggere il codice fino a quando non ne incontra uno; questo è anche il motivo per cui abbiamo potuto usare tre linee per definire la costante `Headline`

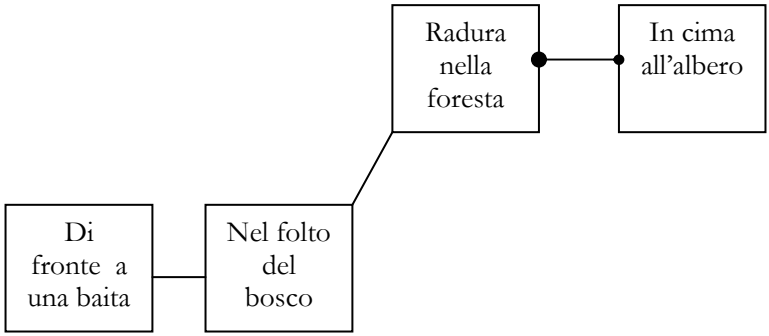
```
Constant Headline
"^Un semplice esempio in Inform
^di Roger Firth and Sonja Kesserich.^";
```

Giusto per ripeterci: ogni gioco che progetterete inizierà da un codice sorgente simile a questo e non sarebbe male tenere una copia di questo modello in un posto sicuro, come punto di partenza per le vostre future avventure. Pensate ad esso come ad una tavolozza di un artista che la prepara con cura per creare i propri capolavori.

Adesso che abbiamo fatto un piccolo tour di Inform, possiamo iniziare a pensare a ciò che richiede in particolare il nostro primo gioco.

Definire le locazioni del gioco

Un buon punto di partenza in ogni gioco è pensare alle locazioni che sono necessarie, perciò andiamo a fare una mappa che mostra le quattro stanze di cui abbiamo bisogno:



Nell' IF, spesso si parla delle locazioni con il termine room (stanza, locazione), anche quando non ci si trova esattamente fra quattro mura. Andiamo a vedere come Inform definisce queste locazioni. Ecco un primo tentativo:

```
Object "Di fronte a una baita"
    with description
           "Ti trovi davanti a una baita. Verso est si stende la
            foresta.",
    has light;

Object "Nel folto del bosco"
    with description
           "Attraverso la folta vegetazione, ti pare di scorgere un
            edificio verso ovest. Un sentiero porta verso
            nord-est.",
    has light;

Object "Una radura nella foresta"
    with description
           "Un alto sicomoro si erge al centro di questa radura.
            Il sentiero si inoltra tra gli alberi, verso sud-
            ovest.",
    has light;

Object "In cima all'albero"
    with description "Ti tieni precariamente appesa al tronco.",
    has light;
```

Ancora una volta possiamo ricavare una serie di principi da questo esempio:

- La definizione di una locazione comincia con la parola `Object` e finisce, quattro linee circa più in basso con un punto e virgola. Ogni componente che appare nel gioco - non solo le locazioni, ma anche le persone, le cose che puoi vedere e toccare, i suoni, gli odori, un vento forte - sono definiti in questa maniera; pensate ad un "oggetto" semplicemente come a un termine generico per la miriade di cose che assieme compongono il modello del mondo che l'avventura riproduce.
- La frase tra virgolette che segue la parola `Object` è il nome che l'interprete usa per fornire al giocatore la lista degli oggetti che circondano il

personaggio protagonista dell'avventura: dove è, cosa può vedere, cosa sta trasportando e così via.

Nota: stiamo usando la parola "giocatore" per indicare sia la persona che sta giocando l'avventura sia il protagonista principale in essa (spesso indicato con il termine PC, Player Character, personaggio protagonista). Dal momento che il PC, ovvero Heidi, è un personaggio femminile, ci riferiremo al giocatore con il termine "Lei" mentre parleremo del gioco.

- Segue la parola chiave `with`, che dice semplicemente al compilatore cosa si deve aspettare dopo.
- La parola `description`, introduce un'altro pezzo di testo che fornisce più dettagli a proposito dell'oggetto: nel caso di una locazione, dà la descrizione di ciò che circonda il giocatore quando entra in quella stanza. La descrizione testuale è racchiusa da virgolette ed è seguita da una virgola.
- Vicino alla fine, appare la parola chiave `has`, che ancora dice al compilatore di aspettarsi un certo tipo d'informazioni.

La parola `light` dice che l'oggetto è una sorgente d'illuminazione, e che quindi il personaggio protagonista può vedere cosa accade nella locazione. Ci dovrebbe essere almeno una sorgente di luce in ogni stanza (a meno che non si voglia gettare il giocatore nell'oscurità); nella maggior parte dei casi viene impostata come sorgente di luce la stessa locazione.

Accenniamo a cosa sono queste parole chiave (ne parleremo più diffusamente nel prossimo capitolo). Un oggetto può avere sia delle **proprietà** (introdotte dalla parola chiave `with`) sia degli **attributi** (che seguono la parola chiave `has`). Una proprietà è caratterizzata da un nome (tipo `description`) e da un valore (come la stringa `"Ti trovi davanti a una baita. Verso est si stende la foresta."`); un attributo invece possiede solamente il nome.

Nel frattempo giocando questa piccola avventura vi accorgerete che inizierà in questo modo:

```
Davanti alla baita
Ti trovi davanti a una baita. Verso est si stende la foresta.
```

Potete quindi vedere come vengono usati il nome della locazione (`Davanti la baita`) e la descrizione (`description`) (`Ti trovi davanti a una baita. Verso est si stende la foresta.`).

Spostarsi tra le locazioni

Abbiamo detto che questo era il primo tentativo di definire le locazioni; è un inizio abbastanza buono, ma mancano ancora diverse informazioni. Se guardate la bozza della mappa, potete vedere come sono disposte le locazioni e come vengono collegate tra di loro; dal "Nel folto del bosco", per esempio, il

giocatore dovrebbe potersi muovere ad ovest verso la baita o a nordest verso la radura. Anche se le descrizioni che abbiamo scritto già menzionano le possibili direzioni, abbiamo bisogno di esplicitare tali movimenti tra le definizioni delle locazioni in una forma che abbia senso per l'interprete. In questo modo:

```
Object davanti_baita "Di fronte a una baita"
    with   description
               "Ti trovi davanti a una baita. Verso est si stende la
                foresta.",
           e_to foresta,
    has light;

Object foresta "Nel folto del bosco"
    with   description
               "Attraverso la folta vegetazione, ti pare di scorgere
                un edificio verso ovest. Un sentiero porta verso
                nord-est.",
           w_to davanti_baita,
           ne_to radura,
    has light;

Object radura "Una radura nella foresta"
    with   description
               "Un alto sicomoro si erge al centro di questa radura.
                Il sentiero si inoltra tra gli alberi, verso sud-
                ovest.",
           sw_to foresta,
           u_to sull_albero,
    has light;

Object sull_albero "In cima all'albero"
    with   description "Ti tieni precariamente appesa al tronco.",
           d_to radura,
    has light;
```

Abbiamo inserito due cambiamenti agli oggetti locazione.

- Come primo punto, tra la parola `Object` e il nome dell'oggetto tra virgolette abbiamo inserito un diverso tipo di nome: un nome identificativo interno che non verrà mai visto dal giocatore, ma che noi potremo usare *all'interno* del sorgente per relazionare tra loro gli oggetti. Nell'esempio, la prima locazione è identificata come `davanti_baita`, e la seconda come `foresta`. Diversamente dal nome esterno contenuto tra le virgolette, il nome identificativo interno deve essere composto da un'unica parola - ovvero senza spazi. Per maggiore facilità di lettura, useremo spesso il carattere underscore (sottolineatura) come pseudo spazio: `davanti_baita` è più chiaro di `davantibaita`.
- Come secondo punto, abbiamo aggiunto, dopo la descrizione della locazione, delle istruzioni che usano i nomi interni identificativi delle locazioni per mostrare come esse siano connesse tra di loro; una istruzione per ogni connessione. L'oggetto `davanti_baita` ha quindi questa linea aggiuntiva:

```
e_to foresta,
```

Essa consente al giocatore che è davanti alla baita di digitare VAI A EST (o EST, o solo E), ed il gioco lo trasporterà alla locazione verso l'identificativo interno specificato, ossia in questo caso verso la `foresta`. Se il giocatore prova a dirigersi verso qualsiasi altra direzione riceverà la risposta "Non puoi andare in quella direzione".

Ciò che abbiamo definito è quindi un tipo di spostamento a senso unico verso est: `davanti_baita` → `foresta`. L'oggetto `foresta` ha altre due istruzioni:

```
w_to davanti_baita,
ne_to radura,
```

La prima linea definisce la direzione verso ovest `foresta` → `davanti_baita` (ciò permette al giocatore di tornare verso la baita), e la seconda definisce la connessione `foresta` → `radura` che va verso nordovest.

Inform prevede 8 direzioni "orizzontali" (`n_to`, `ne_to`, `e_to`, `se_to`, `s_to`, `sw_to`, `w_to`, `nw_to`) due "verticali" (`u_to`, `d_to`) e due speciali, `in_to` e `out_to`. Vedremo l'uso di alcune di queste direzioni nei rimanenti collegamenti tra locazioni.

E' rimasto un ultimo dettaglio prima di poter testare la nostra avventura. Ricorderete che la nostra storia comincia con Heidi che sta davanti alla baita. Dobbiamo quindi dire all'interprete che l'oggetto `davanti_baita` è la locazione da dove il gioco deve iniziare. Per farlo useremo la routine `Initialise`:

```
[ Initialise; location = davanti_baita; ];
```

`location` è una **variabile**, definita dalla libreria, che dice all'interprete in quale locazione risiede il giocatore al momento. Qui stiamo quindi dicendo all'interprete che all'inizio del gioco, il personaggio giocatore è nella stanza `davanti_baita`.

Ora possiamo aggiungere ciò che abbiamo scritto al sorgente del file modello `Heidi.inf`.

A questo punto, dovreste studiare le quattro definizioni delle locazioni, comparandole con la bozza di mappa fatta, fin quando vi sarete convinti di aver capito come si creano le connessioni tra le locazioni.

```
[TYPE]
    !======================================================================
    Constant Story "Heidi";
    Constant Headline
             "^Semplice esempio in Inform
              ^di Roger Firth e Sonja Kesserich.^";
              ! Traduzione di Paolo Lucchesi
    Include "Parser";
    Include "VerbLib";
    Include "Replace";

    !======================================================================
    ! Oggetti di gioco
```

```
Object  davanti_baita "Di fronte a una baita"
    with   description
               "Ti trovi davanti a una baita. Verso est si stende la
                foresta.",
           e_to foresta,
    has light;

Object  foresta "Nel folto del bosco"
    with   description
               "Attraverso la folta vegetazione, ti pare di scorgere
                un edificio verso ovest. Un sentiero porta verso
                nord-est.",
           w_to davanti_baita,
           ne_to radura,
    has light;

Object  radura "Una radura nella foresta"
    with   description
               "Un alto sicomoro si erge al centro di questa radura.
                Il sentiero si inoltra tra gli alberi, verso sud-
                ovest.",
           sw_to foresta,
           u_to sull_albero,
    has light;

Object  sull_albero "In cima all'albero"
    with   description "Ti tieni precariamente appesa al tronco.",
           d_to radura,
    has light;

!=============================================================
! Entry point routines

[ Initialise; location = davanti_baita; ];

!=============================================================
! Grammatica

Include "ItalianG";

!=============================================================
```

Digitate il codice facendo sempre attenzione alla punteggiatura, specialmente alle virgole e ai punti e virgola. Compilate il sorgente e correggete gli eventuali errori che riporta il compilatore. Ora potete giocare l'avventura; naturalmente non potrete fare molto, ma dovreste essere in grado di muovervi liberamente tra le quattro locazioni che avete definito.

NOTA: per minimizzare lo spazio ed il tempo dedicato a scrivere gli esempi, sono state qui usate descrizioni molto brevi, ma potrete naturalmente renderle più lunghe ed interessanti.

Aggiungiamo l'uccello ed il nido

Dati i precedenti, non dovreste essere molto sorpresi nell'apprendere che sia l'uccello che il nido in Inform sono degli oggetti. Cominceremo la loro definizione in questo modo:

```
Object uccello "uccellino"
    with description  "Troppo giovane per saper volare, il
                       passerotto pigola inerme.",
    has ;

Object nido "nido di uccelli"
    with description  "Il nido e' fatto di rametti e sterpi
                       intrecciati.",
    has ;
```

Potete notare come queste definizioni ricalcano lo stesso schema delle locazioni che abbiamo definito precedentemente: una parola unica come identificatore interno (`uccello`, `nido`), e una parola o una frase come nome esterno a beneficio del giocatore (`uccellino`, `nido di uccelli`). Entrambi hanno una descrizione più dettagliata ma, mentre per le stanze questa viene visualizzata quando il giocatore vi entra per la prima volta o quando digita GUARDA, per gli altri oggetti essa viene visualizzata quando il giocatore ESAMINA quell'oggetto. Ciò che gli altri oggetti *non* hanno sono naturalmente le connessioni dei movimenti (`e_to`, `w_to`, etc... che si applicano solo alle locazioni) o la luce `light` (che non è necessaria visto che le locazioni assicurano che vi sia luce sempre disponibile).

Quando il gioco è in esecuzione, il giocatore vorrà poter interagire con questi due nuovi oggetti, dicendo ad esempio ESAMINA L'UCCELLINO o PRENDI il NIDO. Per fare in modo che queste azioni funzionino correttamente dobbiamo specificare la parola (o le parole) con cui il giocatore potrebbe riferirsi ai due oggetti. Il nostro obiettivo è la flessibilità, ossia prevedere il vocabolario del giocatore, in modo di avere buone possibilità che i comandi da lui digitati vengano compresi. Aggiungiamo una riga per ogni definizione:

```
Object uccello "uccellino"
    with   description   "Troppo giovane per saper volare, il
                         passerotto pigola inerme.",
           name   'uccello' 'uccellino' 'passero' 'passerotto'
                  'volatile',
    has ;

Object nido "nido di uccelli"
    with   description   "Il nido e' fatto di rametti e sterpi
                         intrecciati.",
           name   'nido' 'rametti' 'sterpi',
    has ;
```

La parola chiave `name` introduce una linea di virgolette singole `'...'`. Ognuno dei vocaboli quotati rappresenta una **parola del dizionario**, notate che stiamo parlando di una "parola" (`'uccello' 'uccellino'` oppure `'passerotto'`), non di una "frase"; non si possono usare spazi, virgole o periodi nelle parole del dizionario, sebbene vi sia uno spazio *tra* ognuna di esse e l'intera lista finisca con

una virgola. L'idea è che l'interprete decide a quale oggetto il giocatore si riferisce controllando l'intero dizionario delle parole. Se il giocatore cita UCCELLO, o UCCELLINO o PASSEROTTO, è l'`uccello` ciò a cui vuole riferirsi; se invece menziona NIDO, STERPI o RAMETTI si sta chiaramente riferendo al `nido`. E se invece digita PASSEROTTO NIDO o STERPI UCCELLO, l'interprete risponderà educatamente che non capisce un tubo di cosa gli dice il giocatore.

> NOTA: come si è detto all'interno degli apici è possibile inserire una sola parola. Nella versione in inglese originale di questo esempio ci si può riferire al nido dell'uccello anche con il termine `"bird's nest"` formato dalle parole `'bird^s'` e `'nest'`. Notate come abbiamo posto il simbolo `^` all'interno degli apici invece di un apice stesso. Ciò perché altrimenti il compilatore penserebbe che l'apice all'interno della parola indichi invece la fine della stessa. Il simbolo `^` posto all'interno di apici pertanto rappresenta in realtà un apostrofo. Fortunatamente l'italiano non presenta frequentemente casi di questo tipo, a meno di non scrivere in dialetto.

Potreste chiedervi il motivo per cui abbiamo bisogno di una lista di parole `name` per l'uccello e il suo nido, dal momento che li abbiamo già messi nelle locazioni. È perché il giocatore non può interagire con una locazione allo stesso modo con cui interagisce con gli altri oggetti; per esempio, non ha bisogno di scrivere ESAMINA LA FORESTA - gli basta scrivere GUARDA.

La definizione dell'uccello è completa, ma ci sono ancora alcune particolarità da inserire per il nido: abbiamo infatti bisogno di poter mettere l'uccello al suo interno. Possiamo fare ciò definendo il nido come un `container` - ossia un oggetto capace di contenere altri oggetti al suo interno - così il giocatore può digitare METTI (o INSERISCI) L'UCCELLO NEL NIDO. Inoltre definiamo il nido come `open` - aperto - in modo da evitare che l'interprete ci chieda di aprirlo quando gli ordiniamo di mettere l'uccello nel nido.

```
Object nido "nido di uccelli"
    with   description "Il nido e' fatto di rametti e sterpi
           intrecciati.",
           name 'nido' 'rametti' 'sterpi',
    has container open;
```

Ora che abbiamo definito per bene i due oggetti possiamo incorporarli nel gioco. Per farlo dobbiamo scegliere la locazione dove il giocatore può trovarli. Mettiamo l'uccello nella foresta e il nido nella radura. Ecco come fare:

```
[TYPE]
    Object uccello "uccellino" foresta
        with   description   "Troppo giovane per saper volare, il
                              passerotto pigola inerme.",
               name   'uccello'   'uccellino'   'passero'   'passerotto'
                      'volatile',
        has ;

    Object nido "nido di uccelli" radura
        with   description   "Il nido e' fatto di rametti e sterpi
                              intrecciati.",
```

```
        name   'nido' 'rametti' 'sterpi',
    has container open;
```

La prima riga si può leggere come: "Questa è la definizione di un oggetto che viene identificato in questo file come `uccello`, ed è conosciuto dal giocatore con il nome di `uccellino`, esso è inizialmente collocato all'interno dell'oggetto identificato in questo file come `foresta`."

Dove mettere questo nuovo oggetto nel nostro file sorgente? Più o meno dove volete, ma troverete conveniente inserirlo di seguito alla locazione nella quale si trova. Ciò significa aggiungere l'uccello subito dopo la foresta, ed il nido appena dopo la radura. Ecco la parte intermedia del nostro sorgente:

```
!=============================================================
! Oggetti di gioco

Object  davanti_baita "Di fronte a una baita"
    with   description
               "Ti trovi davanti a una baita. Verso est si stende la
                foresta.",
           e_to foresta,
    has light;

Object  foresta "Nel folto del bosco"
    with   description
               "Attraverso la folta vegetazione, ti pare di scorgere
                un edificio verso ovest. Un sentiero porta verso
                nord-est.",
           w_to davanti_baita,
           ne_to radura,
    has light;

Object  uccello "uccellino" foresta
    with   description
               "Troppo giovane per saper volare, il passerotto
                pigola inerme.",
           name   'uccello'   'uccellino'   'passero'   'passerotto'
                  'volatile',
    has   ;

Object  radura "Una radura nella foresta"
    with   description
               "Un alto sicomoro si erge al centro di questa radura.
                Il sentiero si inoltra tra gli alberi, verso sud-
                ovest.",
           sw_to foresta,
           u_to sull_albero,
    has light;

Object  nido "nido di uccelli" radura
    with   description
               "Il nido e' fatto di rametti e sterpi intrecciati.",
           name 'nido' 'rametti' 'sterpi',
    has container open;

Object  sull_albero "In cima all'albero"
    with   description "Ti tieni precariamente appesa al tronco.",
           d_to radura,
    has light;

!=============================================================
```

Apportate i cambiamenti, ricompilate il gioco e provate l'avventura, ecco ciò che dovreste leggere:

```
Nel folto del bosco
Attraverso la folta vegetazione, ti pare di scorgere un edificio
verso ovest. Un sentiero porta verso nord-est.

Puoi vedere un uccellino qui.

>
```

Aggiungiamo l'albero e il ramo

La descrizione della radura menziona un albero fieramente alto, sul quale si suppone che il giocatore potrà arrampicarsi. Andiamo a definire questo nuovo oggetto.

```
[TYPE]
    Object albero "albero di sicomoro" radura
        with   description
                   "Fieramente alto nel mezzo della radura, l'albero
                    sembra essere molto facile da scalare.",
               name 'albero' 'sicomoro' 'tronco',
        has scenery;
```

Ogni istruzione a questo punto dovrebbe risultare familiare, con l'eccezione dell'attributo `scenery` verso la fine. Abbiamo già citato l'albero nella descrizione della radura, così non vogliamo che l'interprete aggiunga una nuova riga "Puoi vedere un albero di sicomoro qui" dopo di essa, come invece avviene per l'uccello ed il nido. Definendo l'albero come `scenery` sopprimiamo tale ripetizione, inoltre impediamo al giocatore la possibilità di raccogliere l'oggetto albero.

Ci rimane da definire un ultimo oggetto: il ramo in cima all'albero. Non ci dovrebbero essere sorprese nella sua definizione:

```
[TYPE]
    Object ramo "ramo largo e robusto" sull_albero
     with description "E' abbastanza largo da sostenere un oggetto.",
          name        'ramo' 'largo' 'robusto',
     has static supporter;
```

Le uniche nuove istruzioni sono due attributi. `static` è simile a `scenery`: previene che il ramo possa essere raccolto dal giocatore, ma *non* sopprime la frase che menziona l'oggetto quando viene data la descrizione della locazione. Invece `supporter` è più simile all'istruzione `container` che abbiamo usato per il nido, con l'eccezione che questa volta il personaggio giocatore può mettere gli oggetti *sopra* il ramo. (Approfittiamo dell'occasione per sottolineare che un oggetto non può normalmente essere allo stesso tempo un `container` *e* un `supporter`). Ecco qui i nostri oggetti:

```
!=============================================================
! Oggetti di gioco

Object  davanti_baita "Di fronte a una baita"
    with   description
               "Ti trovi davanti a una baita. Verso est si stende la
                foresta.",
           e_to foresta,
    has light;

Object  foresta "Nel folto del bosco"
    with   description
               "Attraverso la folta vegetazione, ti pare di scorgere
                un edificio verso ovest. Un sentiero porta verso
                nord-est.",
        w_to davanti_baita,
        ne_to radura,
    has light;

Object  uccello "uccellino" foresta
    with   description
               "Troppo giovane per saper volare, il passerotto
                pigola inerme.",
           name   'uccello'   'uccellino'   'passero'   'passerotto'
                  'volatile',
    has   ;

Object  radura "Una radura nella foresta"
    with   description
               "Un alto sicomoro si erge al centro di questa radura.
                Il sentiero si inoltra tra gli alberi, verso sud-
                ovest.",
           sw_to foresta,
           u_to sull_albero,
    has light;

Object  nido "nido di uccelli" radura
    with   description
               "Il nido e' fatto di rametti e sterpi intrecciati.",
           name   'nido' 'rametti' 'sterpi',
    has container open;

Object  albero "albero di sicomoro" radura
    with  description
               "Fieramente alto nel mezzo della radura, l'albero
                sembra molto facile da scalare.",
           name 'albero' 'sicomoro' 'tronco',
    has scenery;

Object  sull_albero "In cima all'albero"
    with   description "Ti tieni precariamente appesa al tronco.",
           d_to radura,
    has light;

Object  ramo "ramo largo e robusto" sull_albero
    with   description
           "E' abbastanza largo da sostenere un'oggetto.",
        name 'ramo' 'largo' 'robusto',
    has static supporter;

!=============================================================
```

Ancora una volta, apportate i cambiamenti, e cercate magari di fare qualche esperimento su cosa può fare il vostro modello di mondo.

Ritocchi finali

Il nostro primo passo verso la creazione di un'avventura è quasi completato; aggiungiamo solo due ulteriori modifiche. La prima è semplice: Heidi non può arrampicarsi sull'albero tenendo in mano l'uccello ed il nido separatamente: vogliamo che il giocatore metta prima il passerotto nel suo nido.

Un semplice modo di raggiungere questo obiettivo è aggiungere un comando all'inizio del file:

```
[TYPE]
    !=====================================================================
    Constant Story "Heidi";
    Constant Headline
             "^Semplice esempio in Inform
              ^di Roger Firth e Sonja Kesserich.^";
              ! Traduzione di Paolo Lucchesi
    Constant MAX_CARRIED 1;
```

Il valore di `MAX_CARRIED` limita il numero di oggetti che il personaggio protagonista può portare con se contemporaneamente; impostando il suo valore a 1 stiamo dicendo che il giocatore può scegliere di portare o l'uccello o il nido, ma non entrambi. Naturalmente, il limite ignora il contenuto di un oggetto `container` o `supporter` , quindi il nido con l'uccello dentro è considerato come un oggetto singolo.

La seconda modifica è leggermente più complessa ed importante: al momento non esiste un modo di vincere il gioco! L'obiettivo per il giocatore è mettere l'uccello nel nido, portare il nido sull'albero e piazzarlo sul ramo; quando ha compiuto tutte queste azioni, il gioco dovrebbe terminare. Ecco come possiamo fare:

```
[TYPE]
    Object ramo "ramo largo e robusto" sull_albero
        with   description
                   "E' abbastanza largo da sostenere un oggetto.",
               name 'ramo' 'largo' 'robusto',
               each_turn [; if (nido in ramo) deadflag = 2; ],
        has static supporter;
```

NOTA: diamo una piccolo spiegazione di ciò che avviene. Se trovate difficoltà a comprendere questo passaggio non vi preoccupate. È il passaggio più difficile, ed introduce diversi nuovi concetti tutti in una volta. Più avanti nella guida, spiegheremo questi concetti più chiaramente, così se volete potete saltare questo passaggio ora, per riprenderlo più tardi potete farlo tranquillamente.

La variabile `deadflag`, parte della libreria, è normalmente pari a 0. Se si imposta il suo valore a 2, l'interprete noterà il cambiamento e terminerà il gioco con il messaggio "HAI VINTO!". L'istruzione:

```
if (nido in ramo) deadflag = 2;
```

dovrebbe essere letta come: "Controlla se il `nido` è al momento nel ramo (se il ramo è un `container`) o su di esso (se il ramo e' un `supporter`); se la condizione è vera allora imposta la variabile `deadflag` a 2; Altrimenti non fare nulla". La parte che contiene l'istruzione:

```
each_turn [; ... ],
```

dovrebbe essere letta come: "Alla fine di ogni turno (quando il giocatore è nella stessa locazione del ramo) fai qualsiasi cosa sia scritta all'interno delle parentesi quadre". Quindi mettendo il tutto assieme:

- Alla fine di ogni turno (dopo che il giocatore digita qualcosa e preme il pulsante INVIO, e l'interprete ha fatto ciò che gli è stato richiesto) l'interprete controlla se il giocatore e il `ramo` sono nella stessa locazione. Se la condizione non si verifica non accade nulla. Se invece sono insieme, controlla dove è il `nido`. Inizialmente è nella `radura`, quindi non accade nulla.
- Alla fine di ogni turno l'interprete controlla anche il valore della variabile `deadflag`. Normalmente è pari a 0, così non accade niente.
- Finalmente il giocatore mette il `nido` sul `ramo`. "Aha!" dice l'interprete (a se stesso naturalmente), e imposta il valore della `deadflag` a 2.
- Immediatamente dopo, (un'altra parte de) l'interprete la controlla e trova che il valore `deadflag` è cambiato a 2, il che significa che il gioco è stato completato con successo; allora, dice al giocatore "HAI VINTO!".

Il che era ciò che volevamo ottenere per adesso da questo esempio. Apportate questi ultimi cambiamenti ricompilate e provate l'avventura. Probabilmente troverete che alcune cose potevano essere fatte meglio - anche un semplice gioco come questo può essere considerato una possibilità per migliorare - perciò nel prossimo capitolo rivisiteremo un pochino Heidi e la sua foresta. Ma prima ricapitoleremo ciò che abbiamo imparato.

4. Riepiloghiamo le basi

Nel precedente capitolo, la progettazione del nostro primo gioco ha introdotto moltissimi concetti di questo linguaggio, senza però fornirvi molti dettagli su ciò che stava accadendo. Andiamo quindi a rivedere alcune delle cose che abbiamo imparato, cercando di presentarle in maniera più elegante. Parleremo di "Costanti e Variabili", "Definizione degli Oggetti", di "Relazioni tra Oggetti" e de "l'albero degli oggetti", del "Testo quotato" ed infine di "Routine e Istruzioni".

Constanti e variabili

Superficialmente possono apparire simili, ma in realtà sono due argomenti assai differenti.

Costanti

Una **constante** è un nome al quale viene assegnato un valore una ed una volta soltanto e non si può usare in seguito lo stesso nome per un valore differente. Pensate ad essa come ad una lastra di pietra sulla quale incidete un numero: una incisione non può essere cancellata, quindi ogni volta che guarderete la pietra leggerete sempre lo stesso numero. Nel nostro esempio abbiamo visto sia una `Constant` a cui è stato associata come valore una stringa di caratteri:

```
Constant Story "Heidi";
```

Sia una costante a cui è stato associato un numero:

```
Constant MAX_CARRIED 1;
```

Questi due esempi rappresentano i modi più comuni in Inform per usare le costanti.

Variabili

Una **variabile** è un nome a cui viene assegnato un valore, ma tale valore può essere cambiato in qualsiasi momento. Pensate ad essa come ad una lavagna sulla quale segnare un numero con il gesso: quando se ne ha bisogno si può sempre cancellarlo e riscriverne uno nuovo. Non abbiamo impostato nessuna variabile nel nostro primo gioco, ma ne abbiamo utilizzate un paio create dalla libreria:

```
Global location;
Global deadflag;
```

Il valore di una **variabile globale** creata in questa maniera è inizialmente pari a 0, ma si può cambiare in qualsiasi momento. Per esempio, abbiamo usato l'istruzione:

```
location = davanti_baita;
```

per reimpostare il valore della variabile `location` nell'oggetto `davanti_baita`, e abbiamo scritto:

```
if (nido in ramo) deadflag = 2;
```

per reimpostare il valore della variabile `deadflag` a 2.

Più tardi parleremo delle **variabili locali** (si veda il capitolo dedicato alle "Routine") e dell'uso delle proprietà degli oggetti come variabili (si veda il capitolo dedicato agli "Oggetti").

Definizione degli oggetti

Il concetto più importante che dovreste aver appreso dal precedente capitolo è che tutti i giochi sono interamente costruiti su di una serie di oggetti. Ogni locazione è un oggetto, ogni cosa che il giocatore può vedere e toccare è un oggetto; anche il personaggio giocatore stesso è un oggetto (uno di quelli definiti automaticamente dalla libreria). Il modello generale per definire un **oggetto** è:

```
Object id_oggetto "nome_esterno" id_oggetto_padre
    with   proprietà valore,
           proprietà valore,
           ...
           proprietà valore,
    has    attributo attributo ... attributo
    ;
```

La definizione inizia con la parola `Object` e finisce con un punto e virgola; tra i due vi sono tre blocchi principali di informazioni:

- immediatamente dopo la parola `Object` vi sono le informazioni dell'intestazione dell'oggetto;
- la parola `with` introduce le proprietà dell'oggetto;
- la parola `has` introduce gli attributi dell'oggetto.

Intestazione dell'Oggetto

L'intestazione di un oggetto è composta da tre elementi, tutti facoltativi:

- Un identificatore interno `id_oggetto` usato dagli altri oggetti per riferirsi a questo oggetto. È una singola parola (anche se può contenere trattini e underscore), può arrivare fino a 32 caratteri e deve essere unico all'interno del gioco. Si può omettere l'`id_oggetto` se nessun altro oggetto deve relazionarsi con esso.

  Esempio: `uccello`, `albero`, `sull_albero`.

- Un `nome_esterno`, tra virgolette, usato dall'interprete quando si riferisce all'oggetto. Può essere composto da una o più parole e non deve necessariamente essere unico (si potrebbero quindi avere anche più locazioni chiamate `"Da qualche parte nel deserto"`). Sebbene non sia obbligatorio è meglio dare un `nome_esterno` a tutti gli oggetti.

  Esempio: `"uccellino"`, `"albero di sicomoro"`, `"in cima all'albero"`.

- L'`id_oggetto`, ovvero l'identificatore interno di un altro oggetto che è la locazione iniziale dove il nostro oggetto è presente all'inizio del gioco (il suo "genitore" o "padre" – si veda la prossima sezione.). Può essere omesso per gli oggetti che non hanno un genitore all'inizio; viene *sempre* omesso nelle locazioni.

  Esempio: la definizione dell'`uccello` comincia specificando che all'inizio esso può essere trovato nella locazione `foresta` (sebbene più tardi il personaggio giocatore lo raccoglierà e potrà portarlo altrove):

  ```
  Object uccello "uccellino" foresta
      ...
  ```

  L'`albero` comincia allo stesso modo; l'unica differenza reale è che, poiché il giocatore non può muovere un oggetto con l'attributo `scenery`, esso rimarrà sempre nella radura:

  ```
  Object albero "albero di sicomoro" radura
  ...
  ```

  NOTA: c'è un metodo alternativo per definire la locazione iniziale di un oggetto: usando le "frecce" invece che il nome identificativo interno del genitore dell'oggetto (`id_oggetto` o `obj_id`). Ad esempio la definizione dell'`uccello` poteva cominciare in questa maniera:

  ```
  Object -> uccello "uccellino"
  ...
  ```

  Non useremo il metodo delle frecce in questa guida, anche se descriveremo come funziona nel capitolo "Impostare l'albero degli oggetti" nel Capitolo 14.

Le proprietà dell'Oggetto

La definizione delle proprietà di un oggetto sono introdotte dalla parola chiave `with`. Un oggetto può possedere un numero qualsiasi di proprietà, e non c'è bisogno di seguire un ordine particolare. Ogni definizione è composta di due parti: un nome e un valore divisi da uno spazio e con una virgola alla fine.

Pensate alle proprietà come a delle variabili specificatamente associate ad un oggetto. Il valore iniziale della variabile è quello fornito nella definizione della proprietà, ma se necessario esso può essere modificato durante la partita (anche se normalmente la maggior parte delle proprietà non cambia valore).

Ecco alcuni esempi di proprietà che abbiamo incontrato:

```
description "Il nido e' fatto di rametti e sterpi intrecciati.",
e_to foresta,
name 'uccello' 'uccellino' 'passero' 'passerotto' 'volatile',
each_turn [; if (nido in ramo) deadflag = 2; ],
```

Per una fortunata coincidenza questi esempi mostrano anche i diversi tipi di valori che può assumere una proprietà. Il valore associato alla proprietà `description` in questo esempio particolare è una stringa di caratteri tra virgolette; il valore associato alla proprietà `e_to` è il nome identificativo interno di un oggetto; la proprietà `name` è un poco insolita - il suo valore è una serie di parole del dizionario, ognuna posta tra apici; la proprietà `each_turn` ha come valore una **routine incorporata**[7] (vedi "Routine incorporata" più avanti in questo capitolo). L'unico altro tipo di valore che si può trovare comunemente associato ad una proprietà è un semplice numero. Per esempio:

```
capacity 10,
```

In totale la libreria definisce circa 48 proprietà standard – come `name` e `each_turn` - che possono essere associate ai vostri oggetti; una lista completa delle "Proprietà degli Oggetti" è presente nell'appendice F. E in "Guglielmo Tell", nel capitolo 8, mostriamo come inventarsi una proprietà personalizzata.

Attributi degli Oggetti

Un elenco di attributi per un oggetto è introdotto dalla parola chiave `has`. Un oggetto può avere un numero qualsiasi di attributi, che possono essere scritti in qualsiasi ordine e divisi da uno spazio tra ognuno di loro.

Come per le proprietà, potete pensare ad ogni attributo come ad una variabile che è specificatamente associata ad un oggetto. Naturalmente un attributo è

[7] Embedded Routine (NdT).

molto più limitato di una proprietà dal momento che può assumere solo due valori: presente o assente (o anche acceso/spento, pieno/vuoto, vero/falso – una variabile che può assumere solo due valori viene spesso chiamata **flag**, ovvero bandiera). Inizialmente un attributo o è presente (se menzionate il suo nome nella lista di attributi dell'oggetto) o è assente (nel caso inverso); se necessario, il suo valore può cambiare durante la partita (la cosa di solito accade frequentemente). Spesso diremo che un certo oggetto al momento *ha* un certo attributo o che altrimenti *non lo ha*.

Gli attributi che abbiamo incontrato fino adesso sono:

```
container light open scenery static supporter
```

Ognuno di essi risponde ad una domanda: Questo oggetto può contenerne altri? È una fonte di luce? E così via. Se l'attributo è presente allora la risposta è SÌ; se invece non è presente la risposta è NO.

In totale la libreria definisce circa 30 attributi standard che potete associare ai vostri oggetti; nell'Appendice F potete trovare una lista completa degli attributi degli oggetti.

Rapporti tra oggetti – l'albero degli oggetti

Il vostro gioco non è composto solamente di oggetti, ma è fatto anche di relazioni tra essi; Inform fa molta attenzione a tali rapporti e ne monitora costantemente l'evoluzione. Con "rapporti" naturalmente non intendiamo certo dire solo che Walter è il figlio di Guglielmo, mentre Helga e Guglielmo sono semplici amici, ma stiamo parlando di un'operazione costante di registrazione di dove si trova esattamente ogni oggetto rispetto agli altri nel gioco.

Nonostante ciò che abbiamo appena scritto, Inform gestisce le *relazioni* tra gli oggetti in termini di oggetti **genitori** (parent) e oggetti **figli** (child), anche se non nel senso esatto dei termini, come si potrebbe pensare per Walter e Guglielmo.

Quando il personaggio giocatore è in una particolare locazione - ad esempio nella foresta - possiamo dire che:

- L'oggetto foresta è *il* genitore (parent) dell'oggetto giocatore,

o in alternativa che :

- L'oggetto giocatore è *un* figlio (child) dell'oggetto foresta.

Inoltre, se il giocatore sta trasportando un oggetto, ad esempio il nido, possiamo dire che:

- L'oggetto giocatore è *il* genitore (parent) dell'oggetto nido, o che
- L'oggetto nido è *un* figlio (child) dell'oggetto giocatore.

Notate che un oggetto ha *un solo* genitore (parent) o nessun genitore (parent), ma può avere un *qualsiasi numero* di oggetti figli (child) o nessuno. Per fare un esempio di un oggetto che ha più oggetti figli (child), pensate a come abbiamo definito gli oggetti nido e albero:

```
Object nido "nido di uccelli " radura
...
Object albero "albero di sicomoro" radura
...
```

Ricorderete poi che abbiamo usato la terza voce nell'intestazione dell'oggetto per affermare che la radura è il genitore (parent) dell'oggetto nido, e anche che la radura è il genitore dell'oggetto albero, quindi sia il nido che l'albero sono oggetti figli (child) dell'oggetto radura.

> NOTA: una "stanza" perciò non ha niente di magico rispetto agli altri oggetti, è semplicemente un oggetto che non ha *mai* un genitore (parent), e che di tanto in tanto può avere come oggetto figlio (child) il giocatore.

Quando abbiamo definito l'uccello, lo abbiamo posto nella foresta in questa maniera:

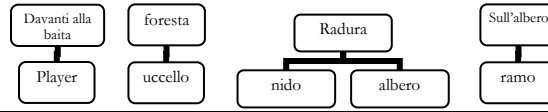
```
Object uccello "uccellino" foresta
 ...
```

Non abbiamo posto nessun altro oggetto in quella locazione, così all'inizio del gioco la foresta è il genitore (parent) dell'uccello e quest'ultimo è l'unico oggetto figlio (child) della foresta. Ma cosa accade quando il personaggio giocatore, che inizialmente si trova nella locazione `davanti_baita`, va verso EST ed entra nella foresta? Risposta: il genitore dell'oggetto giocatore diventa la foresta, così quest'ultima si ritrova con due oggetti figli (child) – l'uccello *ed* il giocatore. Questo è il principio chiave del sistema con cui Inform gestisce i suoi oggetti: i rapporti tra oggetti genitori - figli (parent - child) cambia continuamente, spesso drammaticamente, nello svolgersi del gioco.

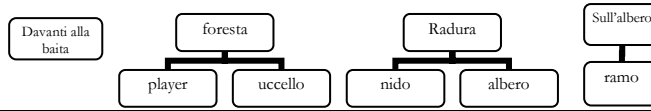
Facciamo un altro esempio: supponiamo che il giocatore prenda l'uccello. Ciò causa un'ulteriore cambio di rapporti tra gli oggetti. L'uccello e' ora un figlio (child) del giocatore (e *non* della foresta), e il giocatore è sia genitore dell'oggetto uccello sia figlio dell'oggetto foresta .

Nel successivo diagramma, mostriamo come i rapporti tra gli oggetti cambino durante il corso di una partita. Le linee verticali rappresentano le relazioni tra genitori - figli (parent - child) con in alto l'oggetto genitore (parent) ed in basso l'oggetto figlio (child); player e' l'oggetto giocatore.

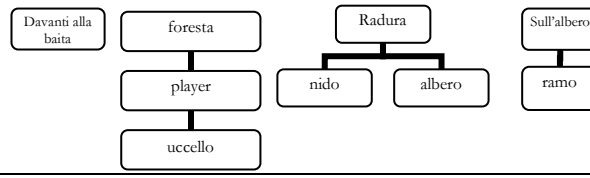
1. All'inizio del gioco



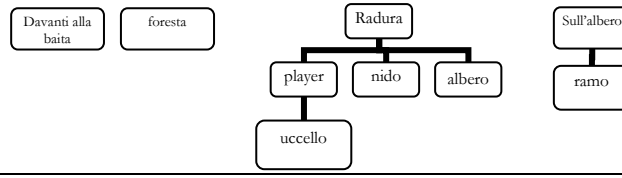
2. Il giocatore scrive VAI A EST :



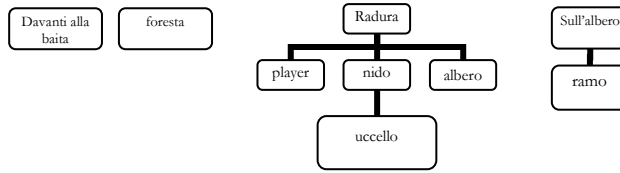
3. il giocatore scrive PRENDI L'UCCELLO:



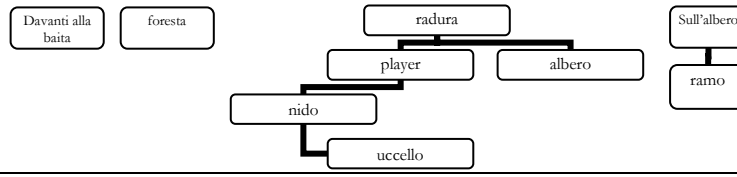
4. il giocatore scrive VAI A NORDEST



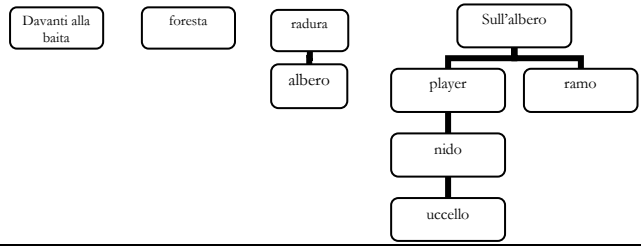
5. il giocatore scrive METTI L'UCCELLO NEL NIDO:



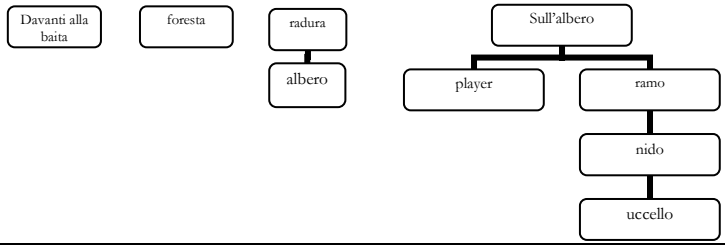
6. il giocatore scrive PRENDI IL NIDO:



7. il giocatore scrive SU:



8. il giocatore scrive METTI IL NIDO SUL RAMO:



In questo breve esempio, ci siamo presi molto tempo e spazio per spiegare esattamente come funziona il modello di rapporti tra gli oggetti - generalmente conosciuto anche con il nome di **albero degli oggetti** (object tree) - e a come appare e cambia a mano a mano che il gioco prosegue. Normalmente non dovrete preoccuparvi di tutti questi dettagli (a) perché l'interprete fa la maggior parte del lavoro al posto vostro e (b) perché nelle avventure testuali ci sono di solito troppi oggetti per poter tracciarne tutti i movimenti. La cosa importante è che capiate i principi di base: in qualsiasi momento un oggetto o non ha un genitore (il che probabilmente vuol dire che è una locazione, o che sta fluttuando nel vuoto e non è al momento facente parte del gioco) o ha un solo ed unico genitore (parent) – ossia l'oggetto "nel quale" o "sul quale" si trova o "di cui fa parte". Naturalmente non c'è limite al numero di figli (children) che un oggetto può avere.

L'uso pratico dei rapporti tra gli oggetti sarà spiegato dettagliatamente più avanti. Programmando la vostra avventure potrete riferirvi al genitore corrente (parent) o ai figli (children) di ogni oggetto con le routine `parent`, `child` e `children`, e questa sarà una delle caratteristiche che userete più frequentemente.

Ci sono anche altre routine associate all'albero degli oggetti (object tree), per aiutarvi a tenere traccia degli oggetti e dei loro movimenti. Le vedremo una per una nei prossimi capitoli. Per un piccolo sommario, si guardi il Capitolo 15 al paragrafo "Oggetti".

Il testo tra virgolette e apici

Inform fa una netta distinzione tra il testo quotato tra virgolette (`"..."`) e quello messo tra apici (`'...'`).

Testo quotato tra virgolette.

Le virgolette vengono usate per contenere una **stringa** – una lettera, una parola, un paragrafo o un qualsiasi numero di caratteri alfanumerici – che si vuole visualizzare sullo schermo attraverso l'interprete durante la partita.

È possibile usare la tilde ~ per rappresentare le virgolette all'interno della stringa, e il carattere ^ per creare una **nuova linea** (ovvero andare a capo). Le lettere maiuscole e minuscole vengono riconosciute come caratteri differenti.

Una stringa lunga può essere suddivisa su più righe; Inform trasforma ogni ritorno a capo nel codice (e ogni spazio intorno al ritorno a capo) in un singolo spazio bianco (ulteriori spazi però verranno preservati). Pertanto queste due stringhe sono equivalenti:

```
"Questa e' una stringa di caratteri."


"Questa
  e'
    una    stringa
                di caratteri."
```

Quando un interprete visualizza una stringa lunga di caratteri – per esempio, mentre descrive una lunga e dettagliata descrizione di una locazione – impiega un sistema automatico di ritorno a capo per visualizzare correttamente il testo sullo schermo del giocatore. Per forzare i ritorni a capo potete utilizzare il carattere ^ , è ciò farà sì che il testo sarà visualizzato in una serie di paragrafi.

Poco fa abbiamo visto delle stringhe utilizzate come valore di una constante:

```
Constant Headline
"^Un semplice esempio in inform
^di Roger Firth e Sonja Kesserich.^";
```

che potrebbero allo stesso modo essere definite così:

```
Constant Headline
"^Un semplice esempio in inform^di Roger Firth e Sonja
 Kesserich.^";
```

e come il valore della proprietà `description` di un oggetto:

```
description    "Troppo giovane per saper volare, il passerotto
                pigola inerme.",
```

Più in la troverete che il loro uso è molto comune nelle istruzioni `print` di stampa.

Testo tra apici

Gli apici sono usati attorno ad una **parola del dizionario**. Questa deve essere una singola parola - senza spazi - che generalmente contiene unicamente lettere (e solo occasionalmente numeri e trattini), Tuttavia è possibile usare il carattere ^ all'interno degli apici per rappresentare un apostrofo.

Le lettere maiuscole o minuscole vengono considerate identiche; inoltre, l'interprete di solito guarda unicamente ai primi nove caratteri di ogni parola che digita il giocatore. Quando il giocatore digita un comando, l'interprete divide il comando in singole parole, che compara con il dizionario. Se trova tutte le parole, ed esse sembrano rappresentare un'azione valida, esso la riconosce ed esegue le opportune istruzioni.

Poco fa abbiamo visto le parole del dizionario usate come valore della proprietà `name` di un oggetto:

```
name 'uccello' 'uccellino' 'passero' 'passerotto' 'volatile',
```

e in effetti questo è praticamente l'unico posto dove il testo tra apici è necessario.

Eviterete ogni confusione al riguardo ricordandovi semplicemente questa distinzione: Output tra Virgolette, Input tra Apici (OVIA).

Routine e istruzioni

Una routine è una serie di istruzioni, che sono studiate (o come diremo spesso, eseguite) durante l'esecuzione del programma dall'interprete. Ci sono due tipi di routine, e circa due dozzine d'istruzioni, potete trovare la loro lista completa nel capitolo 14, o anche andare a leggere "II linguaggio INFORM" nell'appendice E.

Istruzioni

Un'**istruzione** è un comando che dice all'interprete di eseguire una particolare operazione mentre si gioca la partita. Un gioco reale normalmente contiene una moltitudine d'istruzioni, ma per adesso noi ne abbiamo incontrate solo alcune, in particolare:

```
location = davanti_baita;
```

che è un esempio d'**assegnazione**, così chiamata dal momento che utilizza l'operatore di uguaglianza per assegnare un nuovo valore (il nome interno della nostra locazione `davanti_baita`) alla variabile globale `location` che è parte della libreria.

Poi abbiamo incontrato la linea:

```
if (nido in ramo) deadflag = 2;
```

che in realtà contiene *due* istruzioni: un'assegnazione, preceduta da una condizione `if` :

```
if (nido in ramo) ...
```

L'istruzione `if` controlla una particolare condizione; se la condizione è vera, l'interprete esegue qualunque istruzione che la segue, se invece non è vera allora l'interprete ignora l'istruzione successiva. In questo esempio l'interprete sta controllando se l'oggetto `nido` è "dentro" o "sopra" (che noi sappiamo significare "è figlio de") l'oggetto `ramo`. Per la maggior parte della durata del gioco questa condizione non è vera, quindi l'interprete ignora l'istruzione successiva. Quando poi, eventualmente, la condizione diventa vera l'interprete esegue l'istruzione successiva, che prevede una assegnazione:

```
deadflag = 2;
```

che cambia il valore della variabile della libreria `deadflag` dal valore corrente a 2. A proposito dell'istruzione `if`, essa è spesso scritta su due linee, con l'istruzione "controllata" evidenziata da un rientro del margine. Ciò rende il codice più semplice da leggere, e non cambia in alcun modo il suo funzionamento:

```
if (nest in branch)
        deadflag = 2;
```

La condizione che viene controllata dall'istruzione `if` non deve essere una assegnazione ma può essere un qualsiasi tipo di istruzione. Infatti può accadere di avere molte istruzioni, e non solo una, controllate da un'istruzione `if`. Parleremo di queste altre possibilità più avanti nella guida. Per adesso vi basti ricordare che l'unico posto dove potrete trovare delle istruzioni è all'interno di routine indipendenti o incorporate.

Routine indipendenti

Una **routine indipendente** (standalone) è una serie di istruzioni messe assieme e con un nome. Quando la routine viene "chiamata" – attraverso il nome datole - vengono eseguite le istruzioni che contiene. Vediamone una che abbiamo definito nel nostro esempio:

```
[ Initialise; location = davanti_baita; ];
```

Visto che è una routine davvero minuscola, l'avevamo posta su di una singola riga. Andiamola però a riscrivere usando più linee (come con l'istruzione `if`, ciò migliora la leggibilità e non cambia il funzionamento del codice):

```
[ Initialise;
   location = davanti_baita;
];
```

Il primo pezzo `[ Initialise;` è l'inizio della routine, e definisce il nome con cui essa può essere chiamata. La chiusura `];` è la fine della routine. Tra i due vi sono le istruzioni - talvolta esse vengono chiamate il corpo della routine - che sono eseguite quando la routine viene chiamata. Potreste chiedervi: come si fa a chiamare una determinata routine? Presto detto, con un'istruzione di questo tipo:

```
Initialise();
```

Il nome della routine seguita dall'apertura e chiusura di parentesi tonde è tutto ciò che serve per chiamare una routine. Quando incontra una linea come questa, l'interprete esegue le istruzioni - che in questo esempio si limitano ad un'unica assegnazione, ma che potrebbero essere dieci venti o centinaia - contenute nel corpo della routine. Una volta che le ha eseguite, l'interprete ricomincia il suo lavoro dalla linea successiva alla chiamata della routine `Initialise();`.

> NOTA: potreste aver notato che, sebbene abbiamo definito la routine chiamata `Initialise`, nel nostro esempio non l'abbiamo mai "chiamata". Non preoccupatevi - la routine è chiamata dalla libreria di Inform, proprio all'inizio di ogni partita.

Routine incorporate

Una **routine incorporata** (embedded) assomiglia molto ad una routine indipendente, sebbene non abbia un nome e non finisca con un punto e virgola. Eccone una di quelle che abbiamo definito:

```
[; if (nido in ramo) deadflag = 2; ]
```

con l'eccezione del fatto che noi non la abbiamo scritta così ma come un valore di una proprietà dell'oggetto:

```
each_turn [; if (nido in ramo) deadflag = 2; ],
```

che funziona ugualmente anche se scritta in questo modo:

```
each_turn [;
    if (nest in branch)
        deadflag = 2;
],
```

Tutte le routine incorporate sono definite così, ossia come valore di una proprietà di un oggetto, ovvero il luogo dove esse sono racchiuse nell'oggetto stesso. I caratteri introduttivi `[;` potrebbero sembrare un po' strani, ma in realtà è la medesima sintassi delle routine indipendenti (standalone), solo senza nome tra la `[` e il `;`.

Per chiamare una routine incorporata (embedded), in modo da eseguire le istruzioni che contiene, il metodo che abbiamo descritto per le routine indipendenti (standalone) non funziona. Infatti una routine incorporata (embedded) non ha un nome, e non ne ha bisogno; essa viene *automaticamente*

chiamata dalla libreria nel momento opportuno, che viene determinato dal ruolo della proprietà di cui è il valore. Nel nostro esempio, ciò avviene alla fine di ogni turno nel quale il personaggio protagonista è nella stessa locazione del ramo. Più avanti, vedremo altri esempi di routine incorporate, ognuna progettata per realizzare un obiettivo appropriato alla proprietà a cui è associata; vedremo inoltre, che è possibile chiamare anche una routine incorporata (embedded), usando una sintassi del tipo `id_oggetto.propieta'()` - in questo esempio avremmo potuto chiamare la routine scrivendo `ramo.each_turn()`.

Troverete più informazioni su questo argomento in "Routine e argomenti" alla fine del Capitolo 5, in "Routine" nel Capitolo 14, e in "La Piazza del Mercato", nel Capitolo 9.

Siamo arrivati alla fine della nostra revisione degli argomenti coperti nel nostro primo gioco. Avremmo avuto molte più cose da dire, ma stiamo cercando di non sovraccaricarvi troppo in questa fase iniziale e le recupereremo più avanti. Ciò che ci piacerebbe che faceste ora è di tornare indietro a dare un'occhiata al codice del gioco in modo d'assicurarvi di aver ben compreso ogni elemento descritto in questo paragrafo.

Fatto? Bene, allora andiamo avanti a sistemare qualche difetto importante che ancora affligge il nostro gioco.

5. Rivediamo Heidi

Anche nella storia più semplice, il giocatore ha la possibilità di tentare delle azioni che voi non avete previsto. Qualche volta ci possono essere strade alternative per affrontare un problema: se non potete essere sicuri su quale approccio sceglierà il giocatore, dovreste seriamente considerare di permettere tutte le possibilità. Qualche volta gli oggetti che create e le descrizioni che fornite possono suggerire al giocatore che fare così-e-così dovrebbe essere possibile e, perciò, dovreste anche permetterlo. La realizzazione base di un gioco è facile: quello per cui serve tempo, e che rende un gioco vasto e complesso, è l'occuparsi di tutte le *altre* cose che il giocatore può pensare di provare.

Tenteremo di illustrare cosa intendiamo evidenziando alcune delle più notevoli deficienze nel nostro primo gioco.

Ascoltare il passerotto

Ecco un frammento di una partita:

```
Nel folto del bosco
Attraverso la folta vegetazione, ti pare di scorgere un edificio
verso ovest. Un sentiero porta verso nord-est.

Puoi vedere un uccellino qui.

>ESAMINA L'UCCELLINO
Troppo giovane per saper volare, il passerotto pigola inerme.

>ASCOLTA L'UCCELLINO
Non senti niente di inaspettato.

>
```

Non troppo furbo, vero? La nostra descrizione richiama specificatamente l'attenzione del giocatore al verso del passerotto, e poi ci si accorge che non abbiamo niente di speciale da dire a proposito del suo inerme pigolio.

La libreria ha una serie di azioni e risposte per ognuno dei verbi definiti nel gioco, così può rispondere alla maggior parte dei comandi del giocatore con un comportamento standard, invece di rimanere scortesemente in silenzio o dire che non capisce cosa intenda il giocatore. "Non senti niente di inaspettato." è la risposta standard della libreria al comando ASCOLTA, sufficientemente buona per rispondere ad un ASCOLTA IL NIDO o ASCOLTA L'ALBERO, ma decisamente inappropriata in questo caso; abbiamo veramente bisogno di sostituirla con una risposta più rilevante per un ASCOLTA L'UCCELLO. Ecco come fare:

```
Object  uccello "uccellino" foresta
    with   description   "Troppo giovane per saper volare, il
                          passerotto pigola inerme.",
           name   'uccello'   'uccellino'   'passero'   'passerotto'
                  'volatile',
           before [;
               Listen:
                   print "Sembra spaventato e bisognoso d'aiuto.^";
                   return true;
           ],
    has   ;
```

Vediamo quello che abbiamo fatto un passo alla volta:

1. Abbiamo aggiunto una nuova proprietà `before` al nostro oggetto `uccello`. L'interprete controllerà la proprietà `before` *prima* di effettuare qualsiasi azione diretta specificatamente a questo oggetto:

   ```
   before [; ... ],
   ```

2. Il valore della proprietà è una routine incorporata, contenente un'etichetta e due istruzioni:

   ```
   Listen:
       print "Sembra spaventato e bisognoso d'aiuto.^";
   return true;
   ```

3. L'etichetta è il nome di un'azione, `Listen` (Ascolta) in questo caso. Stiamo dicendo all'interprete che, se l'azione che state per eseguire sull'uccello è `Listen` (Ascolta), allora deve prima di tutto eseguire queste due istruzioni; se invece è una qualsiasi altra azione, deve comportarsi normalmente. Così, se il giocatore scrive ESAMINA L'UCCELLO, RACCOGLI L'UCCELLO, METTI L'UCCELLO NEL NIDO, COLPISCI L'UCCELLO o ACCAREZZA L'UCCELLO, otterrà il comportamento standard. Se il giocatore scrive ASCOLTA L'UCCELLO, allora quelle due istruzioni vengono eseguite prima che accada qualsiasi altra cosa. Quando facciamo ciò, si dice che stiamo "intercettando" o "catturando" l'azione di ascoltare l'uccello.

4. Le due istruzioni che eseguiamo sono, prima:

   ```
   print "Sembra spaventato e bisognoso d'aiuto.^";
   ```

   che fa sì che l'interprete stampi la stringa racchiusa tra le virgolette; ricordate che il carattere `^` in una stringa rappresenta il ritorno a capo. Dopo eseguiamo:

   ```
   return true;
   ```

   che comunica all'interprete che non deve fare niente altro, perché abbiamo gestito l'azione `Listen` da soli. E il nostro gioco ora si comporta - perfettamente - in questo modo:

   ```
   >ASCOLTA L'UCCELLINO
   Sembra spaventato e bisognoso d'aiuto.

   >
   ```

L'uso dell'istruzione `return true` probabilmente ha bisogno di qualche spiegazione in più. La proprietà `before` di un oggetto intercetta subito all'inizio un'azione diretta verso quell'oggetto, prima che l'interprete inizi a fare qualsiasi cosa. Questo è il punto in cui vengono eseguite le istruzioni nella routine incorporata. Se l'ultima di queste istruzioni è `return true`, allora l'interprete assume che l'azione sia stata gestita dalle istruzioni precedenti, e non c'è rimasto altro da fare: nessuna azione, nessun messaggio, niente. D'altra parte, se l'ultima istruzione è `return false` allora l'interprete continua ad eseguire le azioni standard come se niente fosse stato intercettato. Ogni tanto potrebbe essere quello che volete veramente che succeda, ma non in questo caso. Se avessimo scritto questo:

```
Object  uccello "uccellino" foresta
    with   description   "Troppo giovane per saper volare, il
                          passerotto pigola inerme.",
           name   'uccello'   'uccellino'   'passero'   'passerotto'
                  'volatile',
           before [;
               Listen:
                   print "Sembra spaventato e bisognoso d'aiuto.^";
                   return false;
           ],
    has   ;
```

allora l'interprete avrebbe prima visualizzato la nostra stringa, e poi avrebbe continuato con il comportamento normale, ovvero avrebbe visualizzato la risposta standard:

```
>ASCOLTA L'UCCELLINO
Sembra spaventato e bisognoso d'aiuto.
Non senti niente di inaspettato.

>
```

La tecnica che abbiamo usato qui – intercettare un'azione diretta ad un particolare oggetto in modo da fare qualcosa d'appropriato per esso – è una procedura che useremo ancora molte e molte volte.

Entrare nella baita

All'inizio del gioco il personaggio principale si trova "di fronte a una baita", il che può portare il giocatore a pensare di poter entrare all'interno di essa:

```
Di fronte a una baita
Ti trovi davanti a una baita. Verso est si stende la foresta.

>ENTRA
Non puoi andare da quella parte.

>
```

Ancora una volta questa non è forse la risposta più appropriata, ma è facile da cambiare:

```
[TYPE]
    Object  davanti_baita "Di fronte a una baita"
        with   description
                   "Ti trovi davanti a una baita. Verso est si stende la
                    foresta.",
               e_to foresta,
               in_to  "E' una stupenda giornata, meglio stare
                       all'aperto.",
               cant_go "L'unico sentiero conduce ad est.",
        has light;
```

La proprietà `in_to` normalmente dovrebbe condurre ad un'altra stanza, nello stesso modo in cui la proprietà `e_to` contiene l'identificatore interno dell'oggetto `foresta`. Comunque, se invece fate in modo che il suo valore sia una stringa, l'interprete visualizzerà quella stringa quando il giocatore prova a entrare. Le altre direzioni non specificate, come NORD e SU, dovrebbero sempre fornire la risposta standard "Non puoi andare da quella parte", ma noi possiamo cambiare anche quello, specificando una proprietà `cant_go` il cui valore sia una stringa appropriata. Otteniamo così questo comportamento:

```
Di fronte a una baita
Ti trovi davanti a una baita. Verso est si stende la foresta.

>ENTRA
È una giornata stupenda, meglio stare all'aperto.

>NORD
L'unico sentiero conduce ad est.

>EST
Nel folto del bosco
...
```

Abbiamo un altro problema adesso; siccome non abbiamo implementato un oggetto per rappresentare la baita, un comando perfettamente ragionevole come ESAMINA LA BAITA riceve ovviamente una risposta priva di senso come "Non vedi nulla del genere". Anche questa eventualità è facile da sistemare; possiamo aggiungere un nuovo oggetto `baita`, rendendolo parte della scenografia, proprio come l'albero.

```
    Object  baita "piccola baita" davanti_baita
        with   description   "È piccola e semplice, ma qui tu sei
                              felice.",
               name 'piccola' 'baita' 'case' 'capanna' 'rifugio',
        has scenery female;
```

Questo risolve il problema, ma ci fornisce immediatamente un'altra risposta irragionevole:

```
Di fronte a una baita
Ti trovi davanti a una baita. Verso est si stende la foresta.

>ENTRA NELLA BAITA
Non è qualcosa in cui puoi entrare

>
```

La situazione è simile a quella del problema ASCOLTA L'UCCELLINO e la soluzione che adottiamo è estremamente simile:

```
Object  baita "piccola baita" davanti_baita
    with description "È piccola e semplice, ma qui tu sei felice.",
         name 'piccola' 'baita' 'case' 'capanna' 'rifugio',
         before [;
             Enter:
                 print_ret  "È una giornata stupenda, meglio stare
                             all'aperto.";
         ],
    has scenery female;
```

Usiamo una proprietà `before` per intercettare l'azione `Enter` (Entra) applicata all'oggetto `baita`, così possiamo visualizzare una risposta più appropriata. Questa volta, però, usiamo una sola istruzione invece di due. La sequenza "`stampa` una stringa che finisce con un ritorno a capo e subito dopo fai un `return true`" è usata così di frequente, infatti, che c'è un'istruzione apposta per fare tutto ciò. Questa è l'istruzione `print_ret` (e notate che la stringa *non* ha bisogno del carattere `^` in fondo):

```
print_ret "È una giornata stupenda, meglio stare all'aperto.";
```

funziona esattamente come

```
print "È una giornata stupenda, meglio stare all'aperto.^";
return true;
```

Avremmo potuto usare le forma più breve anche per ASCOLTA L'UCCELLINO, e la useremo da ora in poi.

Scalare l'albero

Nella radura, quando il giocatore ha in mano il nido e sta guardando l'albero, egli dovrebbe scrivere SU. Però è altrettanto probabile che scriva SCALA L'ALBERO (che correntemente fornisce la risposta completamente fuorviante "Non penso si possa raggiungere qualcosa da qui"). Questa è un'altra opportunità per usare una proprietà `before`, ma ora con una differenza:

```
[TYPE]
    Object  albero "albero di sicomoro" radura
        with   description
                   "Fieramente alto nel mezzo della radura, l'albero
                    sembra molto facile da scalare.",
               name 'albero' 'sicomoro' 'tronco',
               before [;
                   Climb:
                       PlayerTo(sull_albero);
                       return true;
               ],
        has scenery;
```

Questa volta, quando intercettiamo l'azione `Climb` (Scala) applicata all'oggetto albero, non lo facciamo per mostrare un messaggio migliore; lo facciamo perché vogliamo muovere il personaggio del giocatore in un'altra stanza, esattamente come se il giocatore avesse scritto SU. Muovere il personaggio del

giocatore è un'operazione abbastanza complicata, ma fortunatamente tutta questa complessità è nascosta: c'è una **routine della libreria** standard per svolgere questo compito, non una routine che abbiamo scritto noi, ma una che viene fornita come parte del sistema Inform.

Ricorderete che, quando abbiamo menzionato per la prima volta le routine (vedete "Routine indipendenti" nel Capitolo 4), abbiamo usato l'esempio di `Initialise()` e abbiamo detto che "tutto quello di cui abbiamo bisogno per chiamare la routine è il nome della routine seguito da una parentesi aperta e una chiusa". Questo è vero per `Initialise()`, ma non è tutto. Per spostare il personaggio, dobbiamo specificare dove vogliamo che vada, e dobbiamo far ciò fornendo l'identificatore interno della stanza di destinazione tra le due parentesi. In altre parole, invece di chiamare soltanto `PlayerTo()`, chiamiamo `PlayerTo(sull_albero)`, e definiamo `sull_albero` come l'**argomento** della routine.

Anche se abbiamo mosso il personaggio in un'altra stanza, siamo sempre nel mezzo dell'azione `Climb` intercettata. Come in precedenza, dobbiamo comunicare all'interprete che abbiamo gestito noi l'azione, e che non vogliamo il messaggio standard. L'istruzione `return true` fa questo, come al solito.

Far cadere gli oggetti dall'albero

In una stanza normale, come la `foresta` o la `radura`, il giocatore può POSARE qualcosa che sta portando, e l'oggetto lasciato cadrà in effetti ai suoi piedi. Semplice, conveniente, prevedibile – eccetto quando il giocatore è in cima all'albero. Quando egli POSA qualcosa, è un po' improbabile che poi se lo trovi accanto; più facilmente l'oggetto cadrebbe nella radura sottostante.

Sembra che dovremo intercettare l'azione `Drop` (Posare), ma non nel modo in cui l'abbiamo fatto fin'ora. Innanzi tutto non vogliamo complicare le definizioni dell'`uccellino`, del `nido` e di ogni altro oggetto che potremmo introdurre: meglio trovare una soluzione più generale che funzioni per ogni oggetto. E in secondo luogo, dobbiamo riconoscere che non tutti gli oggetti sono posabili; il giocatore non può, ad esempio, POSARE IL RAMO.

Il miglior approccio al secondo problema è quello d'intercettare l'azione `Drop` *dopo* che è accaduta, piuttosto che prima. In questo modo facciamo sì che la libreria si occupi degli oggetti che non sono trasportati dal giocatore o non possono essere posati, e interveniamo solo quando l'azione `Drop` è stata svolta con successo. E il miglior approccio al primo problema è quello d'intercettare il comando agendo non oggetto-per-oggetto come abbiamo fatto fino ad ora, ma invece per ogni azione `Drop` che si svolge nella nostra problematica stanza `sull_albero`. Questo è quello che dobbiamo scrivere pertanto è:

```
[TYPE]
    Object  sull_albero "In cima all'albero"
        with   description "Ti tieni precariamente appesa al tronco.",
               d_to radura,
               after [;
                   Drop:
                       move noun to radura;
                       return false;
               ],
        has light;
```

Rivediamola passo dopo passo:

1. Abbiamo aggiunto una nuova proprietà `after` al nostro oggetto `sull_albero`. L'interprete controlla questa proprietà *dopo* aver fatto qualsiasi azione in questa stanza:

   ```
   after [; ... ],
   ```

2. Il valore della proprietà è una routine incorporata, contenente un'etichetta e due istruzioni:

   ```
   Drop:
       move noun to radura;
       return false;
   ```

3. L'etichetta è il nome di un'azione, in questo caso `Drop` (Posa). Stiamo dicendo all'interprete che, se l'azione che ha appena eseguito è `Drop`, allora deve eseguire queste istruzioni prima di dire al giocatore cosa ha fatto; se è una qualsiasi altra azione deve procedere normalmente.

4. Le due istruzioni che eseguiamo sono prima:

   ```
   move noun to radura;
   ```

   che prende l'oggetto che è appena stato mosso dall'oggetto `player` (giocatore) all'oggetto `sull_albero` (da una azione `Drop` eseguita con successo) e lo sposta ancora in modo che suo genitore (parent) sia l'oggetto `radura`. Quel `noun` è una variabile di libreria che contiene sempre l'identificatore interno dell'oggetto verso cui si rivolge l'azione corrente; se il giocatore scrive POSA UCCELLINO, allora `noun` conterrà l'identificatore interno dell'oggetto `uccello`. Poi, eseguiamo:

   ```
   return false;
   ```

   che dice all'interprete che ora può far sapere al giocatore cosa è successo. Ecco il risultato di tutto questo:

   ```
   In cima all'albero
   Ti tieni precariamente appesa al tronco.

   >POSA IL NIDO
   Posato

   >GUARDA
   In cima all'albero
   Ti tieni precariamente appesa al tronco.

   >GIU
   ```

```
Una radura nella foresta
Un alto sicomoro si erge al centro di questa radura.
Il sentiero si inoltra tra gli alberi, verso sud-ovest.

Puoi vedere un nido di uccelli (nel quale c'è un uccellino) qui.

>
```

Ovviamente potreste pensare che il messaggio standard "Posato" non è molto d'aiuto in queste circostanze particolare. Se preferite suggerire al giocatore cosa è realmente successo, potreste usare questa soluzione alternativa:

```
Object  sull_albero "In cima all'albero"
    with   description "Ti tieni precariamente appesa al tronco.",
           d_to radura,
           after [;
               Drop:
                   move noun to radura;
                   print_ret "Posato... nella radura sottostante.";
           ],
    has light;
```

L'istruzione `print_ret` fa due cose per noi: ci mostra un messaggio più esauriente, ed esegue un `return true` per dire all'interprete che non c'è bisogno di far sapere al giocatore cos'è successo – ci abbiamo già pensato noi.

L'uccellino è nel nido?

Il gioco finisce quando il giocatore mette il nido sul ramo. Abbiamo assunto che l'uccellino sia all'interno del nido, ma potrebbe anche non essere così; il giocatore potrebbe aver prima portato su l'uccello, e poi essere tornato indietro per il nido, o viceversa. Sarebbe meglio non interrompere il gioco fino a che non abbiamo controllato che l'uccellino sia effettivamente nel nido. Fortunatamente è cosa facile da farsi:

```
[TYPE]
    Object  ramo "ramo largo e robusto" sull_albero
        with   description   "È abbastanza largo da sostenere
                              un'oggetto.",
               name 'ramo' 'largo' 'robusto',
               each_turn [;
                   if (uccello in nido && nido in ramo) deadflag = 2;
               ],
        has static supporter;
```

L'istruzione che abbiamo esteso:

```
    if (uccello in nido && nido in ramo) deadflag = 2;
```

ora deve essere letta così: "Controlla se l'`uccello` attualmente è nel (o sul) `nido`, *e* se il `nido` è attualmente sul (o nel) `ramo`; se entrambe le cose sono vere, imposta a 2 il valore di `deadflag`; altrimenti non fare niente".

Riassumendo il tutto

Ora dovreste aver chiaro il bisogno di gestire non soltanto le azioni ovvie che vi vengono in mente quando progettate il gioco, ma anche quanti più possibili modi alternativi in cui il giocatore potrà interagire con gli oggetti che gli presentiamo. Alcuni tra questi modi saranno altamente intelligenti, altri totalmente idioti; in ogni caso dovreste provare ad assicurarvi che la risposta del gioco sia almeno consistente, anche se state dicendo al giocatore "mi dispiace, ma questo non puoi farlo".

I nuovi argomenti che abbiamo incontrato qui sono:

Proprietà degli oggetti

Gli oggetti possono avere una proprietà `before` - se questa esiste, l'interprete la controlla *prima* di compiere un'azione che coinvolge quell'oggetto. Similmente potete fornire una proprietà `after`, che l'interprete controllerà *dopo* avere eseguito un'azione, ma prima di comunicare il risultato al giocatore. Sia la proprietà `before` che la proprietà `after` possono essere usate non solo con oggetti tangibili come l'`uccellino`, la `baita` e l'`albero` (nel qual caso intercettano azioni dirette verso quel particolare oggetto) ma anche con le stanze (nel qual caso intercettano azioni dirette a qualsiasi oggetto nella stanza).

Il valore di ogni proprietà `after` o `before` è una routine incorporata. Se questa routine finisce con `return false`, l'interprete continua ad elaborare l'azione che era stata intercettata; se invece finisce con `return true`, l'interprete non fa altro per quell'azione. Combinando queste possibilità, si può integrare il lavoro svolto da un'azione standard con delle proprie istruzioni o la si può sostituire completamente.

In precedenza abbiamo visto le proprietà di connessione, usate con l'identificatore interno della stanza a cui conducevano. In questo capitolo abbiamo mostrato come il loro valore possa anche essere una stringa (che spieghi come mai il movimento in quella direzione non è possibile). Ecco un esempio di entrambe, ed anche della proprietà `cant_go`, che fornisce una spiegazione per ogni connessione non esplicitamente elencata:

```
    e_to foresta,
    in_to "È una stupenda giornata, meglio stare all'aperto.",
    cant_go "L'unico sentiero conduce ad est.",
```

Routine e argomenti

La libreria ha al suo interno un buon numero di routine utili, disponibili per eseguire alcuni compiti comuni se ne avete bisogno; c'è una lista in "Routine di libreria" nell'Appendice F. Abbiamo usato la routine `PlayerTo`, che sposta il personaggio del giocatore dalla sua stanza corrente ad un'altra - non necessariamente adiacente alla prima stanza. Quando chiamiamo `PlayerTo`, dobbiamo dire alla libreria qual è la stanza di destinazione. Facciamo ciò fornendo l'identificatore interno tra parentesi, quindi:

```
PlayerTo(radura);
```

Un valore tra parentesi è chiamato **argomento** della routine. In effetti una routine può avere più di un argomento; se serve, gli argomenti sono separati da virgole. Ad esempio, per spostare il personaggio del giocatore in una stanza *senza* visualizzare la descrizione di quest'ultima, possiamo fornire un secondo argomento:

```
PlayerTo(radura,1);
```

In questo esempio, l'effetto di quell'1 è quello di impedire che la descrizione della `radura` venga visualizzata.

Istruzioni

Abbiamo incontrato diverse nuove istruzioni:

`return true;`

`return false;`

Abbiamo usato queste due istruzioni alla fine delle routine incorporate per controllare cosa avrebbe fatto l'interprete dopo.

`print "stringa";`

`print_ret "stringa";`

L'istruzione `print` visualizza semplicemente la stringa di caratteri rappresentata qui da `stringa`. L'istruzione `print_ret` fa la stessa cosa, e poi fa un ritorno a capo e finalmente esegue un `return true;`

`move` *`obj_id`* `to` *`parent_obj_id`*

L'istruzione `move` risistema l'albero degli oggetti, rendendo *`obj_id`* figlio di *`parent_obj_id`*.

`if (condition && condition) ...`

Abbiamo esteso la semplice istruzione `if` incontrata prima. I caratteri `&&` (da leggersi "and" o "e") sono un singolo operatore usato comunemente per testare più condizioni allo stesso tempo. Significa "se questa condizione è vera, *e* se anche quest'altra condizione è vera *e*...". Esiste anche un operatore `||`, da leggersi "or" o "o" e un operatore ~~, da leggersi "not" che trasforma il vero in falso e viceversa.

NOTA: in aggiunta ci sono anche gli operatori `& | e ~`, ma essi svolgono un compito diverso e sono molto meno comuni. Fate attenzione a non confondervi.

Azioni

Abbiamo parlato parecchio dell'intercettare azioni come `Listen` (ascolta), `Enter` (entra), `Climb` (scala) e `Drop` (posa). Un'**azione** è una rappresentazione generalizzata di qualcosa che deve essere fatto, determinato dal verbo immesso dal giocatore. Per esempio, i verbi ASCOLTA e SENTI sono due modi di dire più o meno la stessa cosa, e così entrambi risultano nella stessa azione, `Listen`. Similmente, comandi come ENTRA NELLA ..., VAI AL ..., SIEDITI SULLA ..., portano tutti all'azione `Enter`, VAI SU ..., SCALA, SALI SUL ..., portano all'azione `Climb`, e POSA, LASCIA, METTI GIU ..., portano tutti all'azione `Drop`. Questo rende la vita molto più semplice allo sviluppatore; anche se Inform definisce parecchie azioni, esse sono molte meno dei modi in cui queste stesse azioni possono essere espresse in Italiano.

Ogni azione è rappresentata internamente da un numero, ed il valore dell'azione corrente è mantenuto in una variabile di libreria chiamata `action`. Altre due variabili sono utili: `noun` contiene l'identificatore interno dell'oggetto verso cui l'azione si rivolge, e `second` contiene l'identificare interno dell'oggetto secondario (se ce n'è uno).

Ecco qualche esempio:

Comando del giocatore	`action`	`noun`	`second`
ASCOLTA	`Listen`	`nothing`	`nothing`
ASCOLTA L'UCCELLINO	`Listen`	`uccello`	`nothing`
PRENDI L'UCCELLINO	`Take`	`uccello`	`nothing`
METTI L'UCCELLO NEL NIDO	`Insert`	`uccello`	`nido`
POSA IL NIDO	`Drop`	`nido`	`nothing`
METTI IL NIDO SUL RAMO	`PutOn`	`nido`	`ramo`

Il valore `nothing` è una costante (come `true` e `false`) che significa che non c'è un oggetto a cui far riferimento. Potete trovare una lista delle azioni della libreria standard suddivisa in "Azioni di gruppo 1", "Azioni di gruppo 2" e "Azioni di gruppo 3", nell'Appendice F.

Abbiamo finalmente raggiunto la fine del nostro primo gioco. In questi tre capitoli vi abbiamo mostrato i principi base su cui quasi tutti i giochi sono fondati, e vi abbiamo introdotto molte delle componenti di cui avrete bisogno per creare dei giochi più interessanti. Vi suggeriamo di dare un'ultima occhiata al codice sorgente (vedi La Storia di "Heidi", Appendice B), e di procedere poi alla prossima fase.

6. Gugliemo Tell: raccontiamone la storia

Questo capitolo (e i tre che seguono) mostrerà come si procede nella creazione di quel gioco ispirato a "Guglielmo Tell" che abbiamo incontrato all'inizio della guida. Molti dei principi saranno gli stessi che abbiamo spiegato mentre creavamo Heidi e la sua foresta, così non perderemo troppo tempo a descrivere il terreno ormai familiare.

"Guglielo Tell" è un gioco più lungo e più complesso, così cercheremo di arrivare il più velocemente possibile ad esaminare le nuove caratteristiche.

Setup Iniziale

Il nostro punto di partenza è abbastanza simile a quello della volta scorsa. Ecco il primo abbozzo del file `Tell.inf`:

```
[TYPE]
    !% -SD
    !=====================================================================

    Constant Story "Guglielmo Tell";
    Constant Headline
             "^Semplice esempio in Inform
              ^di Roger Firth and Sonja Kesserich.^";
              ! Traduzione di Paolo Lucchesi
    Release 2;  Serial "061028"; ! conto delle release pubbliche
    Constant MAX_SCORE = 3;

    Include "Parser";
    Include "VerbLib";
    Include "Replace";

    !=====================================================================
    ! Classi

    !=====================================================================
    ! Oggetti

    !=====================================================================
    ! Oggetti del giocatore

    !=====================================================================
    ! Entry point routines

    [ Initialise;
        location = strada;
        lookmode = 2;       ! in modo VERBOSO
        move arco to player;
        move faretra to player; give faretra worn;
        player.description =
            "Indossi i tradizionali abiti dei montanari svizzeri.";
        print_ret "^^
            Il luogo: Altdorf, nel cantone Svizzero di Uri. Siamo
            nell'anno 1307, e la Svizzera è sotto il governo
            dell'imperatore Alberto di Asburgo. Il governatore locale -
```

```
        il balivo - è il gradasso Hermann Gessler, che ha posto il
        suo cappello su di un palo di legno nel centro della piazza
        principale; chiunque passi attraverso la piazza deve
        inchinarsi a questo odioso simbolo del potere
        imperiale.
        ^^
        Sei arrivato dalla tua baita sui monti, accompagnato dal tuo
        figlio più giovane, per acquistare derrate. Sei un uomo
        fiero e indipendente, una guida e un cacciatore, conosciuto
        sia per la tua abilità come arciere e, forse poco
        saggiamente (visto che i suoi soldati sono ovunque), per
        essere incapace di nascondere il tuo disprezzo per il
        balivo.
        ^^
        E' giorno di mercato: la città è piena di gente proveniente
        dai vicini villaggi.^";
];

!=============================================================
! Grammatica standard e estensioni

Include "ItalianG";

!=============================================================
```

Noterete che abbiamo aggiunto un paio di divisioni extra nel file, per aiutarci a organizzare meglio il materiale che inseriremo in seguito, ma la struttura generale è identica a quella del nostro primo gioco. Evidenziamo velocemente qualche piccola aggiunta:

- Se guardate l'intestazione del gioco, noterete due informazioni aggiuntive: "Versione" e "Numero di Serie" (oltre alle indicazioni sulle librerie usate).

  ```
  Guglielmo Tell
  Semplice esempio in Inform
  di Roger Firth and Sonja Kesserich.
  Versione 3 / Numero di Serie 061028 / Inform v6.31 Libreria 6/11
  Infit Versione 2.5 SD
  ```

  Questi due campi sono scritti automaticamente dal compilatore, che imposta la Versione a 3 e il Numero di Serie alla data odierna (anno/mese/giorno). Comunque, possiamo esplicitamente aggirare questo comportamento usando i comandi `Release` e `Serial`, per tener traccia di differenti versioni del nostro gioco. Probabilmente nel tempo pubblicheremo diversi aggiornamenti al nostro gioco, con ogni versione che risolverà i problemi riscontrati nella versione precedente. Se qualcuno riscontra problemi nel gioco, sarà bene che ci riferisca esattamente quale versione sta usando; così, piuttosto che prendere i valori di default, impostare i nostri propri valori della versione rilasciata sarà senz'altro di aiuto. Quando verrà il momento di rilasciare una nuova versione, tutto quello che dobbiamo fare è commentare le linee precedenti e aggiungerne una nuova subito dopo:

  ```
  !Release 1; Serial "020128"; ! Prima release beta
  ```

```
!Release 2; Serial "020217"; ! Seconda release beta
Release 3; Serial "020315";  ! Versione per concorso If-Library
```

- Implementeremo un semplice sistema per assegnare punti quando il giocatore fa qualcosa di giusto, quindi definiamo il valore massimo:

```
Constant MAX_SCORE = 3;
```

- La routine `Initialise` che abbiamo scritto l'ultima volta conteneva una sola istruzione, per impostare la posizione di partenza del giocatore. Lo faremo anche questa volta, ma faremo anche altre cose.
- Anzitutto assegneremo il valore 2 alla variabile di libreria `lookmode`. Il comportamento standard di Inform per visualizzare le descrizioni delle stanze è NORMALE (brief: la descrizione viene mostrata solo quando la stanza viene visitata per la prima volta) e, cambiando questa variabile, noi la impostiamo a COMPLETA (verbose: la descrizione viene visualizzata ad *ogni* visita). Questa è soprattutto una questione di gusto personale, e in ogni caso non è niente di più che una convenienza; ci evita soltanto di dover ricordare di scrivere LUNGO ogni volta che proviamo il gioco.
- All'inizio del gioco, vogliamo che Guglielmo sia equipaggiato con il suo arco e la sua faretra. Il modo più corretto di far ciò è quello di effettuare i necessari aggiustamenti all'albero degli oggetti con un paio di istruzioni `move` all'interno della routine `Initialise`.

```
move arco to player;
move faretra to player;
```

e inoltre è il metodo più chiaro per piazzare oggetti nell'inventario del giocatore all'inizio dell'avventura.

> NOTA: aspetta! direte voi. Nel Capitolo scorso per rendere un oggetto figlio (child) di un altro oggetto tutto quello che dovevamo fare era definire l'oggetto figlio specificando internamente il suo oggetto genitore alla fine dell'intestazione:
>
> ```
> Object uccello "uccellino" foresta
> ```
>
> Perché non facciamo la stessa cosa con il giocatore? Perché l'oggetto che rappresenta il giocatore è definito dalla libreria (piuttosto che essere parte del nostro gioco), e ha `selfobj` come identificatore interno; `player` è una variabile il cui valore è tale identificatore. Piuttosto che preoccuparsi di tutto ciò, è più facile usare le istruzioni `move`.

C'è da fare un'ulteriore azione legata alla faretra; è un articolo d'abbigliamento e Guglielmo la sta indossando, cosa che è marcata dall'attributo `worn`. Normalmente l'interprete dovrebbe applicare automaticamente tale attributo gestendo comandi come INDOSSA LA FARETRA, ma siccome abbiamo assegnato direttamente la faretra al

giocatore, dobbiamo anche settare l'attributo `worn`. Il comando `give` fa tale lavoro:

```
give faretra worn;
```

(Per resettare un attributo, fra l'altro, si userà l'istruzione `<give faretra ~worn;>` - leggibile come "assegna alla faretra non-indossato"; Inform usa spesso ~ per come "non".)

- Se il giocatore scrive "ESAMINA ME STESSO", l'interprete mostra la proprietà `description` dell'oggetto `player`. Il valore di default è "Hai sempre lo stesso bell'aspetto", ormai un cliché nel mondo dei giochi Inform. È però facile da cambiare non appena realizzate che, siccome le proprietà di un oggetto sono variabili, potete assegnar loro dei nuovi valori, esattamente come assegnate nuovi valori a `location` e `lookmode`. L'unico problema è quello di specificare la giusta sintassi; non potete scrivere semplicemente:

```
description = "Indossi i tradizionali abiti dei montanari
               svizzeri.";
```

perché ci sono dozzine di oggetti nel gioco, ognuno con la sua proprietà `description`; dovrete essere un po' più espliciti. Dovete scrivere

```
player.description =
   "Indossi i tradizionali abiti dei montanari svizzeri.";
```

- Infine, la routine `Initialise` termina con una lunga istruzione `print_ret`. Siccome l'interprete chiama `Initialise` proprio all'inizio del gioco, quello è il punto in cui tutto questo materiale viene stampato, in modo da fungere da preambolo e da dare un contesto alla scena prima che il gioco vero e proprio inizi. Infatti, tutto ciò che volete voglia venir impostato e fatto all'inizio del gioco deve andare nella routine `Initialise`.

Il gioco non verrà compilato in questo stato, poiché contiene riferimenti a oggetti che non abbiamo ancora definito. In ogni caso, non abbiamo intenzione di costruire il gioco a livelli come abbiamo fatto l'ultima volta, ma piuttosto vogliamo costruirlo in frammenti legati logicamente. Per vedere (e, se volete, per scrivere) l'intero sorgente, andate alla storia "Guglielmo Tell", nell'Appendice C.

Classi di Oggetti

Vi ricordate come abbiamo definito le stanze in "Heidi"? Il nostro primo tentativo è iniziato così:

```
Object  davanti_baita "Di fronte a una baita"
   with   description
              "Ti trovi davanti a una baita. Verso est si stende la
               foresta.",
   has light;

Object  foresta "Nel folto del bosco"
   with   description
```

```
                "Attraverso la folta vegetazione, ti pare di scorgere
                 un edificio verso ovest. Un sentiero porta verso
                 nord-est.",
    has light;
...
```

e abbiamo spiegato come più o meno ogni stanza abbia bisogno dell'attributo `light`, altrimenti il giocatore brancolerebbe letteralmente nel buio. È una piccola scocciatura dover specificare lo stesso attributo tutte le volte; sarebbe più pulito se potessimo dire che *tutte* le stanze sono illuminate. Per far ciò possiamo scrivere:

```
Class Room
has light;

Room davanti_baita "Di fronte a una baita"
    with   description
               "Ti trovi davanti a una baita. Verso est si stende la
                foresta.",
    has ;

Room foresta "Nel folto del bosco"
    with   description
               "Attraverso la folta vegetazione, ti pare di scorgere
                un edificio verso ovest. Un sentiero porta verso
                nord-est.",
    has ;
...
```

Abbiamo fatto quattro cose:

1. Abbiamo detto che alcuni oggetti del nostro gioco saranno definiti dalla parola `Room`, piuttosto che dalla più generale parola `Object`. In effetti abbiamo insegnato ad Inform una nuova parola che verrà usata per definire gli oggetti, e che da ora in poi potremo usare come se avesse sempre fatto parte del linguaggio.
2. Abbiamo inoltre detto che ogni oggetto che definiremo con la parola `Room` avrà automaticamente l'attributo `light`.
3. Abbiamo cambiato il modo in cui definiamo i quatto oggetti "stanza", iniziando la loro definizione con la parola `Room`. Il resto della definizione per questo oggetti - l'intestazione, il blocco delle proprietà, il blocco degli attributi e il punto e virgola finale - rimangono gli stessi; con una eccezione:
4. Non dobbiamo specificare ogni volta l'attributo `light`; ogni oggetto `Room` lo avrà automaticamente.

Una **classe** è una famiglia di oggetti strettamente legati, che si comportano nello stesso modo. Ogni proprietà definita per una classe, e ogni attributo definito per una classe, sono automaticamente assegnati ad ogni oggetto che viene specificatamente definito come appartenente a quella classe; questo processo di acquisizione dovuto all'essere membro di una classe è chiamato **ereditarietà**. Nel nostro esempio abbiamo definito una classe `Room` con l'attributo `light`, e dopo abbiamo specificato quattro oggetti, ognuno dei quali era membro di tale

classe, e ognuno dei quali riceveva l'attributo `light` solo per essere membro della classe.

Perché ci siamo dati tanto da fare? Per tre ragioni principali:

- Spostando le parti di definizione comuni dagli oggetti alla classe che tali oggetti condividono, la definizione di tali oggetti diviene più corta e più semplice. Anche se abbiamo cento stanze, dovremo definire l'attributo `light` una sola volta.
- Creando una parola specializzata che identifica la nostra classe di oggetti, rendiamo il nostro file sorgente più facile da leggere. Piuttosto che anonimi `Object` che possono essere qualsiasi cosa, riconosceremo immediatamente che alcuni oggetti sono locazioni (e altri appartengono ad altre classi differenti che creeremo presto).
- Raccogliendo le definizioni comuni in un solo posto, sarà per noi molto più facile apportare modifiche di ampio respiro. Se dobbiamo fare qualche modifica alla definizione di tutte le stanze, modificheremo soltanto la classe `Room`, e tutti i membri di tale classe erediteranno le modifiche.

Per questa ragione, l'uso delle classi è una tecnica incredibilmente potente, più facile di quanto possa sembrare, e vale la pena di impadronirsene. Da ora, definiremo classi di oggetti ogni volta che sarà conveniente (il che vuol dire generalmente quando due o più oggetti devono comportarsi nella stessa maniera).

Vi potreste chiedere: supponiamo che io voglia definire una stanza che, per qualche ragione, non è illuminata; posso continuare ad usare la classe `Room`? Certo che potete:

```
Room cantina "Cantina buia"
    with   description   "La tua torcia mostra soltanto muri
                         coperti da ragnatele.",
    has ~light;
```

Ciò illustra un'altra simpatica caratteristica dell'ereditarietà: la definizione di un oggetto può annullare la definizione di classe. La classe dice `has light`, ma l'oggetto dice `has ~light` (non ha `light`) e l'oggetto vince. La cantina è buia, e il giocatore avrà bisogno di una torcia per vedere cosa si trova all'interno.

In effetti, per ogni oggetto sia il blocco delle proprietà che il blocco degli attributi sono opzionali e possono essere omessi se non c'è niente da specificare. Ora che l'attributo luce è assegnato automaticamente e che non ci sono altri attributi da assegnare, la parola `has` può essere trascurata:

```
[TYPE]
    Class Room
    has light;

    Room davanti_baita "Di fronte a una baita"
        with   description
                    "Ti trovi davanti a una baita. Verso est si stende la
                     foresta.";
```

```
Room foresta "Nel folto del bosco"
    with   description
               "Attraverso la folta vegetazione, ti pare di scorgere
                un edificio verso ovest. Un sentiero porta verso
                nord-est.";
...
```

Noterete come, se un oggetto non ha un blocco di attributi, il punto e virgola che chiude la sua definizione si sposta alla fine della sua ultima proprietà.

Una classe per le scenografie

Useremo la classe `Room` in "Guglielmo Tell", e qualche altra classe. Ecco una classe `Prop` (oggetto di scena), utile per oggetti scenici il cui solo ruolo è quello di star fermi sullo sfondo aspettando soltanto che il giocatore voglia esaminarli:

```
[TYPE]
    Class Prop
        with   before [;
                   Examine: return false;
                   default:
                       print_ret  "Non hai bisogno di preoccuparti ",
                                  (artdi) self, ".";
               ],
        has scenery;
```

Vedete che tutti gli oggetti di questa classe ereditano l'attributo `scenery`, ciò significa che questi oggetti non sono menzionati nella descrizione della stanza. Più interessante è la proprietà `before`, che risulta maggiormente complessa rispetto ciò che abbiamo visto in precedenza. Vi ricorderete che la prima volta che l'abbiamo incontrata aveva questo aspetto:

```
before [;
    Listen:
        print "Sembra spaventato e bisognoso d'aiuto.^";
        return true;
],
```

Lo scopo della `before` originale era quello di intercettare le azioni `Listen`, lasciando in pace ogni altra azione. Il ruolo della `before` nella classe `Prop` è più ampio: deve intercettare (a) le azioni `Examine` (Esamina), e (b) tutte le altre. Se l'azione è `Examine`, allora il `return false` scritto esplicitamente indica che l'azione procede normalmente. Se l'azione è `default` - ovvero una fra *tutte* quelle non esplicitamente elencate - allora viene eseguita l'istruzione `print_ret`, dopodiché l'interprete non fa altro. Così un oggetto `Prop` può essere esaminato, ma *ogni* altra azione rivolta ad esso si risolve in un messaggio di "non preoccuparti".

Questo messaggio è inoltre più elaborato di altri messaggi visti fino ad ora. L'istruzione che lo produce è

```
print_ret "Non hai bisogno di preoccuparti ", (artdi) self, ".";
```

che può essere letta in questo modo:

1. visualizza la stringa "Non hai bisogno di preoccuparti ",

2. visualizza la proposizione articolata derivata da "di" e adatta, per numero e genere, all'oggetto che stiamo trattando (e quindi visualizza "del", o "della", o "dei"...), seguita da uno spazio e il nome dell'oggetto,
3. visualizza un punto,
4. torna a capo ed esci dalla procedura ritornando `true`, come sempre per un'istruzione `print_ret`.

Le cose interessanti che sono dimostrate da questa istruzione sono:

- Le istruzioni `print` e `print_ret` possono visualizzare più di un singolo frammento d'informazione: possono visualizzare una lista di oggetti separati da virgole. L'istruzione termina con un punto e virgola, al solito.
- Oltre che stringhe posso visualizzare anche nomi di oggetti: per fare qualche semplice esempio preso dal nostro primo gioco, `(the) nido` visualizzerebbe "il nido di uccelli", `(The) nido` visualizzerebbe "Il nido di uccelli", `(a) nido` visualizzerebbe "un nido di uccelli", `(artdi) nido` visualizzerebbe "del nido di uccelli", `(inda) nido` visualizzerebbe "nel nido di uccelli" e `(name) nido` visualizzerebbe semplicemente "nido di uccelli". L'uso di una parola tra parentesi, indicando in questo modo come l'interprete dovrebbe visualizzare l'oggetto riferito dall'indicatore che segue, è chiamata una **regola di stampa**.
- C'è una variabile di libreria `self`, veramente conveniente quando si usa una classe, che contiene sempre un indicatore riferito all'oggetto corrente. Usando questa variabile nella nostra istruzione `print_ret`, ci assicuriamo che il messaggio contenga sempre il nome dell'oggetto appropriato.

Vediamo un esempio pratico di tutto ciò; ecco un oggetto `Prop` preso da "Gugliemo Tell":

```
Prop  "porta meridionale"
    with   name 'cancello' 'porta' 'meridionale',
           description   "Nelle mura della città si apre una larga
                          porta. Il pesante cancello di legno ora è
                          aperto.",
...
```

Se il giocatore scrive ESAMINA LA PORTA, vedrà "Nelle mura della..."; se egli scrive CHIUDI LA PORTA, allora la proprietà `before` della porta entrerà in funzione e visualizzerà "Non hai bisogno di preoccuparti della porta meridionale", prendendo il nome dell'oggetto dalla variabile `self`.

Il motivo per cui facciamo tutto questo, piuttosto che creare un semplice oggetto `scenery` come l'`albero` e la `baita` di Heidi, è quello di gestire il verbo ESAMINA per avere maggior realismo, ed allo stesso tempo far capire chiaramente al giocatore che provare altri verbi sarebbe solo uno spreco di tempo.

Una classe per il mobilio

L'ultima classe che creeremo per ora - parleremo nel prossimo capitolo delle classi `Arrow` e `NPC` - è per gli oggetti di tipo mobilio. Se marcate un oggetto con l'attributo `static`, un tentativo di prenderlo darà come risultato la frase "È fisso al suo posto" - accettabile nel caso del ramo di Heidi (che effettivamente possiamo supporre essere parte dell'albero), un po' meno nel caso di oggetti che sono soltanto ingombranti e pesanti. Questa classe `Furniture` potrà talvolta essere più appropriata:

```
[TYPE]
    Class Furniture
        with   before [;
                   Take,Pull,Push,PushDir:
                       print_ret (The) self, " è troppo pesante.";
               ],
        has static supporter;
```

La sua struttura è simile a quella della classe `Prop`: qualche attributo appropriato e una proprietà `before` per intercettare le azioni dirette verso di essa. Ancora una volta visualizziamo un messaggio che è "personalizzato" per l'oggetto interessato usando la regola di stampa `(The) self`. Questa volta intercettiamo quattro azioni; avremmo potuto scrivere la proprietà in questo modo:

```
    before [;
        Take: print_ret (The) self, " è troppo pesante.";
        Pull: print_ret (The) self, " è troppo pesante.";
        Push: print_ret (The) self, " è troppo pesante.";
        PushDir: print_ret (The) self, " è troppo pesante.";
     ],
```

ma visto che vogliamo la stessa riposta ogni volta, è meglio accorpare tutte quelle azioni in una sola lista, separate da virgole. Se ve lo state chiedendo, `PushDir` è l'azione che viene eseguita in risposta a comandi come SPINGI IL TAVOLO VERSO NORD.

Incidentalmente, un altro vantaggio nel definire classi come queste è che probabilmente potrete riusarle nel vostro prossimo gioco.

Ora che qualche definizione di classe è al suo posto, possiamo procedere definendo qualche locazione e qualche oggetto reali. Se desiderate controllare che tutto ciò che avete scritto fino ad ora sia esatto potete consultare il paragrafo "Compilare strada facendo" nell'Appendice C.

7. Guglielmo Tell, i primi anni

Muovendoci con una certa celerità, definiremo le prime due stanze e le popoleremo con vari paesani e mobilio da strada, equipaggeremo Guglielmo con il suo arco e la sua faretra piena di frecce, e introdurremo Helga, l'amichevole fruttivendola.

Definiamo la strada

Questa è la strada, la locazione dove il gioco inizia:

```
[TYPE]
    Room strada "Una strada di Altdorf"
        with   description [;
                   print  "La piccola strada conduce verso nord alla
                           piazza principale. La gente del luogo sta
                           sciamando nella città attraverso la porta a
                           sud, salutando a voce alta, offrendo prodotti
                           in vendita, scambiando notizie, informandosi
                           con incredulità esagerata sui prezzi delle
                           merci esposte dai mercanti, i cui banchi
                           rendono ancora più difficile l'avanzare in
                           mezzo alla folla.^";
               if (self hasnt visited)
                   print  "^~Stammi vicino, figliolo,~ dici,
                           ~altrimenti potresti perderti fra tutta
                           questa gente.~^";
               ],
               n_to vicino_piazza,
               s_to
                   "La folla, muovendosi verso la piazza a nord, ti
                    impedisce di tornare indietro.";
```

Stiamo usando la nostra nuova classe `Room`, così non c'è bisogno dell'attributo `light`. Le proprietà `n_to` e `s_to`, i cui valori sono rispettivamente un identificatore interno e una stringa, sono tecniche che abbiamo usato già in precedenza. L'unica innovazione è che la proprietà `description` ha una routine incorporata nel suo valore. La ragione è la seguente:

La prima cosa che incontriamo nella routine è l'istruzione `print`, che visualizza i dettagli della strada e dell'ambiente circostante. Se questo fosse stato tutto ciò che volevamo fare, avremmo potuto fornire questi particolari usando una stringa come valore della proprietà `description`; questi due esempi, cioè, si comportano in modo identico:

```
    description [;
       print  "La piccola strada conduce verso nord alla piazza
               principale. La gente del luogo sta sciamando nella città
               attraverso la porta a sud, salutando a voce alta,
               offrendo prodotti in vendita, scambiando notizie,
               informandosi con incredulità esagerata sui prezzi delle
               merci esposte dai mercanti, i cui banchi rendono ancora
```

```
            più difficile l'avanzare in mezzo alla folla.^";
],

description
       "La piccola strada conduce verso nord alla piazza
        principale. La gente del luogo sta sciamando nella città
        attraverso la porta a sud, salutando a voce alta, offrendo
        prodotti in vendita, scambiando notizie, informandosi con
        incredulità esagerata sui prezzi delle merci esposte dai
        mercanti, i cui banchi rendono ancora più difficile
        l'avanzare in mezzo alla folla.",
```

Comunque, questo *non è* tutto quello che vogliamo fare. Dopo aver presentato la descrizione di base, vogliamo visualizzare quel piccolo frammento di dialogo in cui Guglielmo dice a suo figlio di essere prudente. E vogliamo che ciò accada una volta sola, la prima volta che la descrizione della strada viene visualizzata. Se il giocatore scrive GUARDA più volte, o va a nord e poi torna verso sud alla strada, noi siamo ben felici di vedere l'ambiente nuovamente descritto, ma non vogliamo che il dialogo si svolga ancora. Questo paio d'istruzioni realizza ciò che vogliamo:

```
if (self hasnt visited)
    print  "^~Stammi vicino, figliolo,~ dici, ~altrimenti potresti
            perderti fra tutta questa gente.~^";
```

Il frammento di dialogo è realizzato dall'istruzione `print`, che è controllata dall'istruzione `if`, che a sua volta esegue il misterioso controllo `self hasnt visited`. Guardiamo più da vicino:

- `visited` è un attributo, ma non uno che normalmente assegnereste voi stessi ad un oggetto. E' assegnato automaticamente agli oggetti stanza dall'interprete, ma solo dopo che la stanza è stata visitata per la prima volta dal giocatore.
- `hasnt` (e `has`) sono disponibili per verificare se un certo attributo è stato assegnato o meno ad un certo oggetto. `X has Y` è vera se e solo se l'oggetto `X` possiede attualmente l'attributo `Y`, e falsa altrimenti. Per invertire il controllo, `X hasnt Y` è vera se e solo se l'oggetto `X` attualmente non possiede l'attributo `Y`, e falsa se invece lo possiede.
- `self`, che abbiamo già incontrato nel capitolo scorso, è un'utilissima variabile che, all'interno di un oggetto, si riferisce sempre a tale oggetto. Siccome la stiamo usando all'interno dell'oggetto strada, si riferirà proprio alla strada.

Così, riassumendo il tutto, `self hasnt visited` risulta vera (e quindi l'istruzione `print` viene eseguita) solo quando l'oggetto strada *non* ha l'attributo `visited`. Siccome l'interprete assegna ad una stanza l'attributo `visited` appena il giocatore c'è stato almeno una volta, questa condizione sarà vera solo per un turno. Quindi, il frammento di dialogo sarà visualizzato solo durante un singolo turno: la prima volta che il giocatore si trova in strada, all'inizio del gioco.

Anche se la variabile `self` risulta di primaria importanza nella definizione delle classi, può essere conveniente usarla semplicemente all'interno di un oggetto. Perché? potrete chiedere. Perché non scrivere l'istruzione in questo modo:

```
if (strada hasnt visited)
    print  "^~Stammi vicino, figliolo,~ dici,
            ~altrimenti potresti perderti fra tutta questa
            gente.~^";
```

È vero che l'effetto è identico, ma ci sono un paio di buoni motivi per usare `self`. Primo: è un aiuto per comprendere il codice anche giorni o settimane dopo che l'avete scritto. Se leggete la riga `if (street hasnt visited)`, dovete pensare per un momento a qual è l'oggetto a cui si riferisce la condizione; oh, è proprio questo. Se leggete `if(self hasnt visited)`, *saprete* immediatamente di quale oggetto stiamo parlando.

Un'altra ragione è l'auto-plagio. Molte volte vi accorgerete che un frammento di codice è utile in differenti situazioni (ad esempio, se volete ripetere il meccanismo della descrizione della strada in un'altra locazione). Piuttosto che riscrivere tutto da capo, userete il tipico copia-incolla per ripetere la routine, e allora tutto quello che dovrete fare sarà scrivere le appropriate stringhe descrittive per la nuova stanza. Se avete usato `self`, la riga `if (self hasnt visited)` è sempre buona; se invece avete scritto `if (strada hasnt visited)`, dovete cambiare anche quella. (Peggio ancora, se vi *dimenticate* di cambiarla il gioco funzionerà ancora, ma non nel modo che voi intendevate, e il bug risulterebbe parecchio difficile da individuare).

Aggiungiamo un po' di scenografia

La descrizione della strada menziona diversi oggetti - la porta, la gente, etc. etc. - che meritano di esistere all'interno del gioco (almeno in una minima forma) per sostenere l'illusione della marea di folla in movimento. La nostra classe `Prop` è l'ideale per questo:

```
[TYPE]
    Prop "porta meridionale" strada
        with   name 'cancello' 'porta' 'meridionale',
               description
                   "Nelle mura della citt@`a si apre una larga porta. Il
                    pesante cancello di legno ora @`e aperto.";

    Prop "banchi"
        with   name 'banchi',
               description
                   "Cibo, abiti, attrezzature da montagna; la solita
                    roba.",
               found_in strada vicino_piazza,
        has pluralname;
```

```
Prop "frutti"
    with   name   'beni' 'merci' 'frutti' 'cibo' 'abiti' 'montagna'
                  'attrezzature' 'cose' 'roba',
           description
                  "Niente che possa attirare la tua attenzione.",
           found_in strada vicino_piazza,
has pluralname;

Prop "mercanti"
    with   name 'mercante' 'mercanti',
           description
               "Qualche truffatore, ma per il resto gente a posto
                che si guadagna da vivere.",
           found_in strada vicino_piazza,
    has animate pluralname;

Prop "gente"
    with   name 'persone' 'gente',
           description "Gente di montagna, come te.",
           found_in [; return true; ],
    has animate female;
```

Nota: dal momento che non ci riferiremo a questi oggetti in altre parti del codice, abbiamo trascurato di dare loro nome interno di identificazione. In ogni caso gli *abbiamo* dato un nome esterno dal momento che fanno parte della classe `Prop` che usa l'istruzione `print_ret ... (the) self`.

Potete vedere un paio di nuovi attributi: `animate` individua un oggetto come "essere vivente", mentre `pluralname` specifica che il nome di un oggetto è plurale, piuttosto che singolare. L'interprete usa questi attributi per assicurarsi che i messaggi relativi a tali oggetti siano grammaticalmente appropriati (per esempio, si riferirà d'ora in poi a "dei mercanti" piuttosto che a "un mercanti"). Siccome la libreria gestisce molte situazioni automaticamente, è difficile stabilire esattamente quale messaggio verrà generato in risposta alle azioni del giocatore; il miglior approccio è quello di restare sul sicuro e assegnare sempre ad un oggetto un insieme pertinente di attributi anche quando, come in questo caso, probabilmente non serviranno.

Vedete anche la nuova proprietà `found_in`, che specifica la locazione o le locazioni dove questo oggetto può essere trovato. Per la porta della città, che appare solo in una stanza, avremmo potuto facilmente includere l'identificatore dell'oggetto genitore (parent) nella riga d'intestazione:

```
Prop "porta meridionale" strada
    with   name 'cancello' 'porta' 'meridionale',
           description   "Nelle mura della città si apre una larga
                          porta. Il pesante cancello di legno ora è
                          aperto.",
    has female;
```

ma abbiamo scelto di usare `found_in` per uniformarci agli altri oggetti scenici, che sono più interessanti. I mercanti e i loro banchi possono essere ESAMINATI sia nella strada che vicino alla piazza - e sono resi come un unico oggetto che l'interprete sposta in una di queste stanze quando il giocatore si trova lì. E la popolazione locale è ancora più onnipresente.

La proprietà `found_in` di un oggetto è una routine incorporata, e quello che dovrebbe generalmente fare è controllare la locazione corrente e restituire `true` se vuole che l'oggetto in questione sia presente in quella locazione, e `false` altrimenti. Qui noi vogliamo che la gente sia *sempre* presente, in ogni stanza, così restituiamo `true` senza nemmeno preoccuparci di esaminare la locazione. È come se avessimo scritto una di queste altre possibilità, ma in modo più semplice e con meno possibilità d'errore:

```
Prop "gente"
    with   name 'persone' 'gente',
           description "Gente di montagna, come te.",
           found_in   strada vicino_piazza piazza_sud centro_piazza
                      piazza_nord mercato,
    has animate female;
```

```
Prop "gente"
    with   name 'persone' 'gente',
           description "Gente di montagna, come te.",
           found_in [;
               if (location==strada || location==vicino_piazza ||
                  location==piazza_sud || location==centro_piazza ||
                  location==piazza_nord || location==mercato)
                          return true;
               return false;
               ],
    has animate female;
```

```
Prop    "gente"
    with   name 'persone' 'gente',
           description "Gente di montagna, come te.",
           found_in [;
               if (location==strada or vicino_piazza or piazza_sud
                      or centro_piazza or piazza_nord or mercato )
                          return true;
              return false;
           ],
    has animate female;
```

Nel secondo esempio vedete l'operatore `||`, che deve essere letto come "o", e che abbiamo menzionato verso la fine di "Heidi"; esso combina i vari confronti `location == qualche_stanza` in modo che tutta l'istruzione `if` risulta vera se *anche uno solo* di questi confronti risulta vero.

E nel terzo esempio introduciamo la parola `or`, che è un modo più succinto per ottenere esattamente lo stesso risultato.

Gli oggetti del giocatore

Visto che la nostra routine `Initialise` li ha già menzionati, vorremmo definire l'arco e le frecce di Guglielmo:

```
[TYPE]
    Object  arco "arco"
        with   name 'arco',
               description "Il tuo fidato arco di tasso.",
               before [;
                   Drop,Give,ThrowAt:
                       print_ret "Non ti separi mai dal tuo arco.";
               ],
        has clothing;

    Object  faretra "faretra"
        with   name 'faretra',
               description
                   "Fatta in pelle di capra, di solito pende dalla tua
                    spalla sinistra.",
               before [;
                   Drop,Give,ThrowAt:
                   print_ret "Ma @`e un regalo di tua moglie Hedwig.";
           ],
        has container open clothing female;
```

Entrambi sono oggetti semplici, che intercettano le azioni `Drop`, `Give` e `ThrowAt` per essere sicuri che Guglielmo non ne sia mai privo. L'attributo `clothing` appare per la prima volta, indicando che sia la faretra, sia (meno probabilmente) l'arco possono essere indossati; ricorderete che la nostra routine `Initialise` assegna l'attributo `worn` alla faretra.

Una faretra vuota è abbastanza inutile, così questa è la classe usata per definire la riserva di frecce di Guglielmo. Questa classe ha qualche caratteristica inusuale:

```
[TYPE]
    Class Arrow
        with   name 'freccia' 'frecce//p',
               article "una",
               plural "frecce",
               description   "Come tutte le altre tue frecce, dritta e
                              acuminata.",
               before [;
                   Drop,Give,ThrowAt:
                       print_ret "Troppo pericolose, meglio non
                                  lasciarle in giro.";
               ];
```

Le classi che abbiamo creato fino ad ora – `Room`, `Prop` e `Furniture` – erano pensate per oggetti che si comportano nella stessa maniera ma sono altrimenti chiaramente separati. Per esempio, un tavolo, un letto e un armadio generalmente avrebbero le loro caratteristiche individuali – un nome, una descrizione, forse qualche proprietà particolare – e allo stesso tempo erediterebbero il comportamento generale degli oggetti `Furniture`. Le frecce non sono così: non soltanto si comportano nello stesso modo, ma sono altresì indistinguibili l'una dall'altra. (Beh, potremmo aspettarci che un esperto arciere

come Guglielmo possa riconoscerle, ma certamente noi non possiamo.) Vogliamo ottenere questo effetto:

```
>INVENTARIO
Stai portando:
    una faretra (indossata)
        tre frecce
    un arco
```

dove l'interprete accorpa insieme la nostra scorta di tre frecce, piuttosto che elencarle individualmente in questa goffa maniera:

```
>INVENTARIO
Stai portando:
    una faretra (indossata)
        una freccia
        una freccia
        una freccia
    un arco
```

L'interprete fa questo per noi se i nostri oggetti sono "indistinguibili", effetto che viene meglio ottenuto rendendoli membri di una classe che contiene sia la proprietà `name` che la proprietà `plural`. Definiamo le frecce reali molto semplicemente, così:

```
[TYPE]
    Arrow   "freccia" faretra;
    Arrow   "freccia" faretra;
    Arrow   "freccia" faretra;
```

e potete vedere che forniamo solo due informazioni per ogni oggetto `Arrow`: un nome esterno tra virgolette ("freccia" in questo caso) che l'interprete usa per riferirsi all'oggetto, e la locazione iniziale (nella faretra). Questo è tutto: niente blocco delle proprietà, niente insieme di attributi, nessun identificatore interno, perché non avremo mai bisogno di riferirci ad un individuale oggetto `Arrow` all'interno del gioco.

La proprietà `name` ha un aspetto bizzarro:

```
    name 'freccia' 'frecce//p'
```

La parola `'freccia'` si riferisce ad una singola freccia. La stessa cosa succederebbe alla parola `'frecce'`, a meno che non specifichiamo all'interprete che è un riferimento plurale. Il suffisso `//p` individua la parola `'frecce'` come potenziale riferimento a più di un oggetto alla volta, permettendo quindi al giocatore di scrivere PRENDI FRECCE e quindi raccogliere tutte le frecce disponibili (senza di esso PRENDI FRECCE avrebbe raccolto solo una freccia a caso).

Ci sono altre due proprietà che ancora non abbiamo visto:

```
    article "una"
    plural "frecce"
```

La proprietà `article` vi permette di definire l'articolo indefinito dell'oggetto - usualmente qualcosa come "un", o "alcune" - invece di lasciare che la libreria ne assegni uno automaticamente. È spesso una precauzione di sicurezza. La

maggior parte degli interpreti assegna i giusti articoli, ma se vogliamo restare sul sicuro, possiamo definire esplicitamente la parola appropriata[8].

E la proprietà `plural` definisce la parola che deve essere usata quando mettiamo insieme diversi di questi oggetti, come in "tre frecce" nell'inventario. L'interprete non può determinare automaticamente il plurale di ogni parola; il plurale di freccia è frecce e non freccie, ed il plurale di "fetta di torta" non è usualmente "fetta di torte", per esempio.

Andiamo avanti lungo la strada

Come Guglielmo si muove verso nord, verso la piazza, arriva in questa stanza:

```
[TYPE]
    Room vicino_piazza "Lungo la strada"
        with   description
                   "La gente continua a premere e a farsi strada dalla
                    porta sud alla piazza principale, che si trova
                    appena più a nord. Riconosci la proprietaria di un
                    banco di frutta e verdura.",
               n_to piazza_sud,
               s_to strada;
```

Nessuna sorpresa qui, e nemmeno nella maggior parte degli oggetti scenici.

```
[TYPE]
    Furniture banco "banco di frutta e verdura" vicino_piazza
        with   name 'frutta' 'verdura' 'banco' 'tavolo',
               description
                   "Davvero un piccolo banco, con un grosso mucchio di
                    patate, qualche carota, qualche rapa, un po' di
                    mele.",
               before [; Search: <<Examine self>>; ],
        has scenery;

    Prop "patate" vicino_piazza
        with   name 'patata' 'patate' 'mucchio',
               description
                   "Devono essere state raccolte un po' di tempo fa...
                    almeno 300 anni!",
        has pluralname female;

    Prop "frutta e verdura" vicino_piazza
        with   name 'carota' 'carote' 'rapa' 'rape' 'mele' 'vegetali',
               description "Prodotti locali.",
        has female;
```

[8] Nella versione inglese la parola "arrow" (freccia), iniziando per vocale, avrebbe bisogno dell'articolo "an" piuttosto che "a". Per questo motivo la proprietà `article` viene introdotta in questo punto. La corrispondente parola italiana "freccia" non ha la stessa particolarità, ma abbiamo lasciato l'introduzione della proprietà `article` in questo punto. Accenniamo qui invece alla presenza di un'altra proprietà denominata `articles` espressamente prevista per le lingue non anglofone che hanno nomi irregolari e che hanno bisogno di diversi articoli davanti a secondo delle circostanze. `articles` permette di specificare tre articoli per il nome irregolare. Maggiori informazioni a pagina 272 del DM4. NdT.

L'unica novità qui è la proprietà `before` nel banco di frutta e verdura. La descrizione del banco - un mucchio di oggetti sul tavolo - può suggerire al giocatore che egli può CERCARE fra i vari prodotti, magari trovando fortunosamente una barbabietola o qualcos'altro di interessante. Il giocatore non è così fortunato, ma noi vogliamo lo stesso permettere questo tentativo. Avendo intercettato l'azione `Search`, il nostro piano è quello di rispondere con la descrizione del banco, come se il giocatore avesse scritto ESAMINA IL BANCO. Non c'è un modo semplice per infilare letteralmente quelle parole all'interno dell'interprete (in modo che CERCA generi un'azione `Examine` e non un'azione `Search`), ma possiamo simulare l'effetto che esse creano: un'azione di `Examine` applicata all'oggetto banco. Questa istruzione abbastanza criptica esegue il giusto compito:

```
<Examine banco>;
```

Avendo deviato l'azione `Search` in un'azione `Examine`, dobbiamo dire all'interprete che non ha bisogno di fare altro, siccome abbiamo già gestito completamente l'azione noi. Abbiamo già fatto questo in precedenza - usando `return true` - e così un primo tentativo per la nostra `before` avrà questo aspetto

```
before [; Search: <Examine banco>; return true; ],
```

La sequenza di due istruzioni `<...>; return true;` è così comune che c'è una singola istruzione che le abbrevia: `<<...>>`. Inoltre, esattamente per le stesse motivazioni che abbiamo mostrato prima, il nostro codice è più chiaro se usiamo `self` invece di `banco`. Così questa è finalmente la nostra proprietà `before`:

```
before [; Search: <<Examine banco>>; ],
```

Un paio di osservazioni finali prima di lasciare quest'argomento. L'esempio qui è di un'azione applicata ad un oggetto (`self`, anche se `banco` o `noun` avrebbero funzionato lo stesso). Potete usare le istruzioni `<...>` e `<<...>>` per azioni che non influiscono su oggetti:

```
<<Look>>;
```

(che rappresenta il comando LOOK), o che influiscono su due oggetti. Per esempio, il comando METTI L'UCCELLINO NEL NIDO può essere simulato da questa istruzione:

```
<<Insert uccello nido>>;
```

Introduciamo Helga

Uno degli aspetti più complicati del realizzare un buon gioco è quello di fornire una interazione soddisfacente con gli altri personaggi. È già abbastanza difficile codificare un oggetto inanimato che generi risposte appropriate a qualunque azione il Personaggio Giocatore (PG, o PC - Player Character) possa provare.

La situazione peggiora quando gli "altri oggetti" sono creature viventi – Personaggi Non Giocatori (PNG, o NPC - Non-Player Character) – con, si suppone, una mente propria. Un buon NPC dovrebbe muoversi indipendentemente, eseguire azioni con uno scopo, iniziare conversazioni, rispondere a ciò che dici o fai (e anche a ciò che *non* dici e non fai): può essere un vero incubo.

Ma non qui: manterremo i nostri tre NPC – Helga, Walter e il balivo – più semplici possibile. Nonostante ciò, possiamo stabilire alcuni principi fondamentali; ecco la classe su cui baseremo i nostri NPC:

```
[TYPE]
    Class   NPC
        with  life [;
                    Answer,Ask,Order,Tell:
                     print_ret "Usa soltanto PARLA [", (arta) self, "].";
              ],
        has animate;
```

La cosa più importante qui è l'attributo `animate` - ciò che definisce un oggetto come NPC e fa sì che l'interprete lo tratti in modo lievemente differente - ad esempio, PRENDI HELGA ha come risposta "Non credo che Helga voglia farsi prendere in braccio".

L'attributo `animate` inoltre, porta in gioco nove azioni extra che possono essere applicate solo a oggetti animati: `Answer` (rispondi), `Ask` (chiedi), `Order` (ordina) e `Tell` (parla) sono tutte associate alla comunicazione vocale, e `Attack` (attacca), `Kiss` (bacia), `Show` (mostra), `ThrowAt` (lancia a) e `WakeOther` (sveglia) sono associati all'interazione non verbale. In più la nuova proprietà `life` – molto simile a `before` - può essere definita per intercettare queste azioni. Qui noi la usiamo per intercettare i comandi verbali come CHIEDI A HELGA DELLE MELE o PARLA A WALTER DEI BAMBINI, rispondendo al giocatore che in questo gioco abbiamo implementato soltanto con un più semplice comando PARLA - vedi "Verbi, verbi, verbi", nel Capitolo 9.

Basata su questa classe, ecco Helga:

```
[TYPE]
    NPC proprietaria "Helga" vicino_piazza
        with   name   'proprietaria' 'verduraia' 'venditrice'
                      'negoziante' 'mercante' 'helga' 'vestito'
                      'sciarpa' 'fruttivendola',
               description
                      "Helga è una donna grassoccia e affabile,
                       infagottata in un abito informe e una sciarpa a
                       pois.",
               initial [;
                  print "Helga smette di sistemare le patate
                              e ti saluta calorosamente.^";
               if (location hasnt visited) {
                  move mela to player;
                  print "^~Salve, Guglielmo, è una buona giornata per
                     il commercio! Questo è il giovane Walter? Come è
                     cresciuto... Ecco, questa è una mela per lui... Se
                     toglie la parte ammaccata, il resto è buono. Come
                     sta la signora Tell? Salutamela davvero...~^";
```

```
            }
        ],
        frasi_dette 0,  ! per contare gli argomenti di conversazione
        life [;
            Kiss: print_ret "~Ooh, che sfacciato!~";
            Talk:  self.frasi_dette = self.frasi_dette + 1;
                   switch (self.frasi_dette) {
                       1:  score = score + 1;
                           print_ret  "Ringrazi calorosamente Helga
                                       per la mela.";
                       2:  print_ret  "~Ci vediamo presto.~";
                       default: return false;
                   }
        ],
    has female proper;
```

Gli attributi sono `female` - in modo che l'interprete si riferisca ad Helga con i pronomi appropriati - e `proper`. Quest'ultimo significa che il nome esterno di questo oggetto è un nome proprio, e così non deve essere preceduto da "una" o "la": non vorrete mica visualizzare "Puoi vedere una Helga qui" o "Non penso che la Helga voglia essere presa in braccio". Potete anche notare come la variabile di libreria `score` sia stata incrementata. Questa variabile custodisce il numero di punti che il giocatore ha segnato; quando cambia come in questo caso, l'interprete informa il giocatore che "Il tuo punteggio è appena aumentato di un punto".

Ci sono anche le proprietà `life` e `frasi_dette` (di cui parleremo in "Guglielmo Tell: la fine è prossima", nel capitolo 9) e una proprietà `initial`.

La proprietà `initial` è usata quando l'interprete sta descrivendo la stanza ed elencando gli oggetti che potete vedere. Se *non* l'avessimo definita, avremmo ottenuto questo:

```
Lungo la strada
La gente continua a premere e a farsi strada dalla porta sud alla
piazza principale, che si trova appena più a nord. Riconosci la
proprietaria di un banco di frutta e verdura.
Puoi vedere Helga qui.

>
```

ma noi vogliamo introdurre Helga in modo più interattivo, ed è per questo motivo che esiste la proprietà `initial`: sostituisce il messaggio standard "Puoi vedere *oggetto* qui" con un messaggio personalizzato di vostra ideazione. Il valore della proprietà `initial` può essere una stringa da visualizzare o, come in questo caso, una routine incorporata. Questa è abbastanza simile alla proprietà `description` che abbiamo definito per la strada: qualcosa che viene visualizzato *sempre* (Helga smette di...) e qualcosa che viene stampato solo alla prima occasione ("Salve, Guglielmo, è una buona giornata..."):

```
Lungo la strada
La gente continua a premere e a farsi strada dalla porta sud alla
piazza principale, che si trova appena più a nord. Riconosci la
proprietaria di un banco di frutta e verdura.
Helga smette di sistemare le patate e ti saluta calorosamente.
"Salve, Guglielmo, è una buona giornata per il commercio! Questo è
il giovane Walter? Come è cresciuto... Ecco, questa è una mela per
lui... Se toglie la parte ammaccata, il resto è buono. Come sta la
signora Tell? Salutamela davvero..."

>
```

Ma non è esattamente identica alla routine `description` della strada. Anzitutto, abbiamo bisogno di un controllo `if` lievemente diverso: `self hasnt visited` funziona bene per un oggetto locazione, ma questa routine fa parte di un oggetto che si trova all'interno di una stanza; invece potremmo usare o `vicino_piazza hasnt visited` o (meglio) `location hasnt visited` - giacché `location` è la variabile di libreria che si riferisce alla stanza in cui il giocatore si trova al momento. E secondariamente, sono aperte alcune parentesi graffe `{...}`: perché?

Alla prima visita di Guglielmo in questa stanza, dobbiamo fare due cose:

- assicurarci che Guglielmo entri in possesso della mela, poiché questo fatto è menzionato mentre...
- visualizzare il festoso saluto di Helga.

L'istruzione `move` esegue il primo compito, mentre l'istruzione `print` esegue il secondo. Ed entrambe le istruzioni sono controllate dall'istruzione `if`.

Fino ad ora, abbiamo usato l'istruzione `if` due volte, in entrambi i casi per controllare solo una singola istruzione.

```
if (nido in ramo) deadflag = 2;
if (self hasnt visited)
    print  "^~Stammi vicino, figliolo,~ dici, ~altrimenti potresti
            perderti fra tutta questa gente.~^";
```

Questo è ciò che fa l'istruzione `if` - controlla se l'istruzione che segue debba essere eseguita oppure no. Così come possiamo controllare contemporaneamente due istruzioni? Bene, *potremmo* scrivere due istruzioni if:

```
if (location hasnt visited) move mela to player;

if (location hasnt visited)
    print  "^~Salve, Guglielmo, è una buona giornata per il
            commercio! Questo è il giovane Walter? Come è
            cresciuto... Ecco, questa è una mela per lui... Se
            toglie la parte ammaccata, il resto è buono. Come sta la
            signora Tell? Salutamela davvero...~^";
```

Ma ciò è incredibilmente rozzo; useremo, invece, le parentesi graffe per raggruppare le istruzioni `move` e `print` in un unico **blocco di istruzioni** (chiamato talvolta blocco di codice, oppure soltanto blocco) che conta come un'unica istruzione per quanto riguarda il controllo datogli dall'istruzione `if`.

```
if (location hasnt visited) {
    move mela to player;
    print  "^~Salve, Guglielmo, è una buona giornata per il
            commercio! Questo è il giovane Walter? Come è
            cresciuto... Ecco, questa è una mela per lui... Se
            toglie la parte ammaccata, il resto è buono. Come sta la
            signora Tell? Salutamela davvero...~^";
}
```

Un blocco può contenere una, due, dieci, cento istruzioni; non fa differenza - saranno tutte trattate come un'unica entità dall'istruzione `if` (e dall'istruzione `objectloop` che incontreremo più avanti, e dalle istruzioni `do`, `for` e `while`, che non incontreremo affatto in questa guida).

NOTA: l'esatta posizione delle graffe è una questione di scelta personale. Noi usiamo questo stile:

```
if (condizione) {
    istruzione;
    istruzione;
    ...
}
```

ma altri autori hanno altre preferenze, tra cui:

```
if (condizione) {
    istruzione;
    istruzione;
    ...
    }

if (condizione)
{
    istruzione;
    istruzione;
    ...
}
if (condizione)

    {
    istruzione;
    istruzione;
    ....
    }
```

Anche se fino ad ora non ne abbiamo avuto bisogno, ora probabilmente è il momento opportuno per introdurre l'estensione `else` all'istruzione `if`. Qualche volta potremmo voler eseguire un blocco di istruzioni se una certa condizione è vera, e un altro blocco di istruzioni se invece non è vera. Ancora una volta *potremmo* scrivere due istruzioni `if`:

```
if (location has visited) {
    istruzione;
    istruzione;
    ...
}
```

```
if (location hasnt visited) {
    istruzione;
    istruzione;
    ...
}
```

Ma questo ben difficilmente può essere reputato un approccio elegante; una clausola `else` svolge il lavoro in modo molto più pulito:

```
if (location has visited) {
    istruzione;
    istruzione;
    ...
}
else {
    istruzione;
    istruzione;
    ...
}
```

Abbiamo creato un bel po' di ambienti e scenografia, ma l'azione vera e propria deve ancora arrivare. Ora arriverà il momento di definire la piazza della città e creare il confronto tra Guglielmo e i soldati del balivo.

8. Gugliemo Tell: il cuore della storia

L'azione del nostro gioco si avvicina al suo apice nella piazza principale della città. In questo capitolo definiremo le stanze che costituiscono la piazza e ce la vedremo con Guglielmo che si avvicina al cappello che si trova sopra il palo - lo saluterà, oppure rimarrà fieramente ribelle?

La parte sud della piazza

La piazza principale, comunemente intesa come un unico enorme spazio aperto, è rappresentata da tre stanze. Questa è la parte sud:

```
[TYPE]
    Room piazza_sud "Lato sud della piazza"
        with   description
                   "La piccola strada che conduce verso sud si apre
                    nella piazza principale, per poi riprendere dalla
                    parte opposta di questo affollato luogo di ritrovo.
                    Per continuare lungo la strada, verso la tua
                    destinazione - la conceria di Johansson - devi
                    attraversare, andando verso nord, la piazza, al
                    centro della quale vedi il cappello di Gessler messo
                    su di un palo. Se vuoi andare avanti, non puoi
                    evitare di passarci vicino. Soldati
                    imperiali si fanno largo rudemente tra la folla,
                    spingendo, calciando e imprecando ad alta voce.",
               n_to centro_piazza,
               s_to vicino_piazza;

    Prop "cappello su un palo"
        with   name 'cappello' 'palo',
               before [;
                   default:
                       print_ret "Sei troppo lontano per il momento.";
               ],
               found_in piazza_sud piazza_nord;

    Prop "soldati di Gessler"
        with
            name 'soldato' 'soldati',
            description
                   "Uomini sgarbati e violenti, non di queste parti.",
            before [;
                FireAt:
                    print_ret "Sono abbondantemente in sovrannumero.";
                Talk:
                    print_ret  "Una  simile  feccia  non  merita  la  tua
                    considerazione.";
            ],
            found_in piazza_sud piazza_nord centro_piazza mercato,
        has animate pluralname;
```

Tutta roba abbastanza standard: solo una stanza (`Room`) e tre oggetti di scenografia (`Prop`). Il "vero" palo si trova nella stanza che rappresenta il centro della piazza, il che vuol dire che il giocatore non può ESAMINARLO da questa

stanza (tecnicamente, non è "in scope"). Comunque, siccome noi reputiamo che Guglielmo possa vedere l'intera piazza da dove si trova, abbiamo bisogno di creare un falso palo e un falso cappello, che possono essere trovati (proprietà `found_in`) sia in questa locazione che nella parte nord della piazza, anche se sono "troppo lontani" per una descrizione dettagliata.

Tra l'altro ci si trova troppo lontano per compiere *qualsiasi* azione con il palo o il cappello. Dal momento che l'avversione di Guglielmo per le attività del governo è un aspetto centrale della storia, un messaggio del tipo "Non hai bisogno di preoccuparti per il cappello" sarebbe un inganno inaccettabile.

Gli odiosi soldati sono anch'essi implementati abbozzandoli appena; devono star lì, ma non devono fare molto altro. La loro caratteristica più interessante è probabilmente il fatto che intercettano due azioni, `FireAT` e `Talk`, che non sono parte della libreria, ma piuttosto sono due nuove azioni che abbiamo definito apposta per questo gioco. Parleremo di queste azioni in "Verbi, Verbi, Verbi", all'interno del Capitolo 10, e a quel punto il ruolo di questa proprietà `before` avrà molto più senso.

Il centro della piazza

Le attività che si svolgono qui sono fondamentali per la trama del gioco. Guglielmo è arrivato dalla parte sud della piazza, e ora incontra il palo su cui è stato posto il cappello. Egli può fare tre cose:

1. Ritornare a sud. Questo è permesso, ma non è altro che una piccola perdita di tempo - non c'è niente di utile a sud.
2. Prestare omaggio al palo, e poi procedere verso nord. Questo è permesso, ma stravolge tutta la storia.
3. Provare ad andare a nord senza salutare il palo. Per due volte un soldato lo fermerà e gli darà un avviso verbale. Al terzo tentativo, la sua pazienza sarà esaurita e Guglielmo verrà portato a fare il suo numero.

Così ci sono due azioni che dobbiamo aspettarci: `Salute` ovvero saluta (intercettata dal palo) e `Go` ovvero vai (che può essere intercettata dalla stessa locazione). `Go` è un'azione standard della libreria. `Salute` è un'azione che noi abbiamo sviluppato; vediamocela con questa, anzitutto. Ecco una prima versione della stanza e del palo:

```
Room centro_piazza "In mezzo alla piazza"
    with   description
               "C'è meno folla al centro della piazza; la maggior
                parte della gente preferisce tenersi il più lontano
                possibile dal palo che troneggia in questo luogo,
                reggendo quell'assurdo cappello cerimoniale. Un
                gruppo di soldati rimane nei pressi, osservando
                chiunque passi di qui.",
           n_to piazza_nord,
           s_to piazza_sud;
```

e il palo col cappello:

```
[TYPE]
    Furniture palo "cappello su un palo" centro_piazza
        with
            name   'palo'  'legno'  'tronco'  'pino'  'cappello'  'nero'
                   'rosso' 'cuoio' 'tesa' 'piume',
            description
                   "Il palo, il tronco di un piccolo pino, solo pochi
                    centimetri di diametro, sar@`a alto tre o quattro
                    metri. Sulla cima @`e stato appoggiato con cura il
                    ridicolo cappello di cuoio nero e rosso di Gessler,
                    dalla larga tesa e adornato con un ciuffo di piume
                    d'oca tinte.",
            salutato false,
            before [;
                FireAt:
                    print_ret "Allettante, ma non sei in cerca di guai.";
                Salute:
                    self.salutato = true;
                    print_ret
                           "Saluti il cappello sull'alto palo.
                            ^^
                            ~Grazie davvero, messere~, sghignazzano i
                            soldati.";
            ],
  has   scenery;
```

La stanza avrà presto bisogno di nuovo lavoro, ma il palo è completo (notate che abbiamo semplificato leggermente la faccenda facendo sì che un solo oggetto rappresentasse sia il palo, sia il cappello che si trova sul palo). Esso menziona una proprietà che non abbiamo ancora incontrato: `salutato`. Che coincidenza: la libreria fornisce una proprietà con un nome perfettamente adatto al nostro gioco, e per di più in italiano; sicuramente no? No, naturalmente no. La proprietà `salutato` non è una proprietà standard della libreria; è una proprietà che noi abbiamo appena inventato. Notate come è stato facile – abbiamo semplicemente incluso la riga:

```
    salutato false,
```

nella definizione dell'oggetto, e voilà, abbiamo aggiunto la nostra proprietà fatta in casa, inizializzandola al valore false. Ora, quando dobbiamo cambiare lo stato della proprietà, possiamo semplicemente scrivere:

```
    palo.salutato = true;
    ...
    palo.salutato = false;
```

o soltanto (all'interno dell'oggetto palo):

```
    self.salutato = true;
    ...
    self.salutato = false;
```

Possiamo anche controllare, se necessario, qual è lo stato attuale della proprietà:

```
    if (palo.salutato == true) ...
```

Notate che usiamo == come test per "è uguale a"; sono due segni d'uguaglianza, e non uno solo come accade per assegnare un valore. Non confondete la doppia uguaglianza (==) con quella singola (=) che serve invece ad assegnare un valore ad una variabile. Confrontate questi esempi:

Corretto	Sbagliato
`Score = 10;`	`Score == 10;`
Assegna il valore 10 a `score`	Non fa nulla, il valore di score rimane inalterato
`If (score==10)`	`If (score=10)`
Esegue la successiva istruzione solo se il valore di `score` è pari a 10	Assegna 10 a `score`, quindi esegue sempre le istruzioni che seguono, l'assegnazione restituisce sempre `true`, quindi il test è *sempre* vero.

Definire una nuova variabile proprietà che, invece che applicarsi a tutti gli oggetti nel gioco (come fanno le proprietà standard della libreria), è specifica solo di una classe di oggetti o anche - come in questo caso - solo di un singolo oggetto, è una tecnica potente e comune. In questo gioco abbiamo bisogno di una variabile vero/falso per indicare se Guglielmo ha salutato il palo o no: il modo migliore è quello di crearne una che sia parte del palo. Così, quando il palo intercetta l'azione `Salute`, noi facciamo due cose: usiamo l'istruzione `self.salutato = true` per registrare il fatto, ed usiamo un'istruzione `print_ret` per dire al giocatore che il saluto è stato "graditamente" ricevuto.

> NOTA: creare una nuova proprietà variabile come questa - all'interno del palo, come in questo caso - è l'approccio raccomandato, ma non è l'unica possibilità. Menzioneremo brevemente alcuni approcci alternativi in "Leggere il codice di altre persone", nel Capitolo 14.

Torniamo al centro della piazza. Come abbiamo già detto, vogliamo individuare i tentativi di Guglielmo di lasciare questa stanza, cosa che possiamo fare intercettando l'azione `Go` nella proprietà `before`. Abbozziamo il codice di cui abbiamo bisogno:

```
before [; Go:
    if (noun == s_obj) { Guglielmo prova ad andare verso sud }
    if (noun == n_obj) { Guglielmo prova ad andare verso nord }
];
```

Possiamo facilmente intercettare l'azione `Go`, ma in che direzione si sta muovendo? Bene, scopriamo che l'interprete trasforma un comando VAI VERSO SUD (o solo SUD, o S) in un'azione `Go` applicata ad un oggetto `s_obj`. Questo oggetto è definito dalla libreria; ma perché non è chiamato semplicemente "`south`"? Beh, perché abbiamo già un altro tipo di sud, la proprietà `s_to` usata per dire cosa si trova in direzione sud quando definiamo

una locazione. Per evitare di confonderle, `s_to` significa "verso sud" e `s_obj` significa "sud quando il giocatore lo scrive come oggetto di un verbo".

L'identità dell'oggetto verso cui è diretta l'azione corrente è conservata nella variabile `noun`, così possiamo scrivere l'istruzione `if (noun == s_obj)` per verificare se il contenuto della variabile `noun` è uguale all'identificatore dell'oggetto `s_obj` - e quindi se Guglielmo sta provando ad andare verso sud. Un'altra istruzione simile controlla se egli sta cercando di andare verso nord, e questo è tutto quello che ci interessa; possiamo lasciare che gli altri movimenti se la sbrighino da soli.

Le parole *Guglielmo prova ad andare verso sud* non fanno parte del nostro gioco; sono solo un'annotazione temporanea che ci ricorda che, se delle istruzioni devono essere eseguite in questa situazione, questo è il punto in cui metterle. In realtà, questo è il caso più semplice; è quando Guglielmo prova ad andare a nord che il divertimento comincia. Dobbiamo scegliere una fra due diverse linee di comportamento, a seconda che egli abbia o meno salutato il palo. Ma noi sappiamo se ciò è successo; la proprietà `salutato` del palo può dircelo. Così espandiamo il nostro abbozzo in questo modo:

```
before [;
    Go:
        if (noun == s_obj) {Guglielmo prova ad andare verso sud [1]}
        if (noun == n_obj) {Guglielmo prova ad andare verso nord...
            if (palo.salutato == true) {
                ...ed ha salutato il palo [2] }
            else {
                ...ma non ha salutato il palo [3] }
        }
    ];
```

Abbiamo un'istruzione `if` annidata dentro un'altra istruzione `if`. E c'è di più: l'istruzione `if` interna ha una clausola `else`, il che significa che possiamo eseguire un blocco di istruzioni se la condizione `if(palo.salutato == true)` è vera, e un altro blocco di istruzioni se invece è falsa. Rileggetela con attenzione, guardando come le graffe si accoppiano; è abbastanza complessa, e avete bisogno di comprendere cosa sta succedendo. Una cosa importante da ricordare è che, a meno che voi non inseriate graffe per cambiare ciò, la clausola `else` si riferirà sempre all'istruzione `if` più recente. Paragonate questi due esempi:

```
if (condizione1) {
    if (condizione2) { sia condizione1 che condizione2 sono vere }
    else { condizione1 è vera e condizione2 è falsa }
}

if (condizione1) {
    if (condizione2) { sia condizione1 che condizione2 sono vere }
}
else { condizione1 è falsa }
```

Nel primo esempio, la clausola `else` si riferisce all'`if` più recente, mentre nel secondo caso la diversa posizione delle graffe porta la clausola `else` a riferirsi all'`if` precedente. Ripensateci finché non comprendete il perché.

NOTA: noi abbiamo usato l'indentazione come guida visiva per indicare come sono in relazione gli `if` e gli `else`. Fate attenzione, però; il compilatore assegna ogni `else` al suo `if` soltanto sulla base del raggruppamento logico (n.d.t.: dato dalle graffe), senza prestar attenzione a com'è disposto il codice.

Torniamo alla proprietà `before`. Dovreste essere in grado di vedere che i casi marcati con `[1]`, `[2]` e `[3]` corrispondono ai tre possibili sviluppi dell'azione che abbiamo elencato all'inizio di questa sezione. Scriviamo il codice per questi, uno alla volta.

Caso 1: Ritornare a sud

Nel primo caso, *Guglielmo cerca di tornare a sud*; non c'è molto da fare:

```
avvertimenti 0,       ! per contare gli avvertimenti dei soldati
    before [;
        Go:
            if (noun == s_obj) {
                self.avvertimenti = 0;
                palo.salutato = false;
            }
            if (noun == n_obj) {
                if (palo.salutato == true)
                  { andando verso nord, dopo aver salutato il palo }
                else
                  { andando verso nord, senza aver salutato il palo }
            }
     ];
```

Guglielmo può arrivare fino al centro della piazza, dare un'occhiata al palo e ritornare prontamente a sud. Oppure può fare uno o due (ma non tre) tentativi per andare a nord, prima, e poi andare a sud. *Oppure* potrebbe essere veramente perverso e salutare il palo per poi tornare verso sud. In tutti questi casi, noi lo riporteremo a zero, come se non avesse ricevuto avvertimenti (indipendentemente da quanti ne ha effettivamente ricevuti) e come se il palo non fosse stato salutato (indipendentemente dal fatto che lo abbia salutato oppure no). In effetti facciamo conto che il soldato abbia davvero la memoria corta, e che si dimentichi completamente di Guglielmo se il nostro eroe si allontana dal palo.

Per fare tutto ciò abbiamo aggiunto una nuova proprietà e due istruzioni. La proprietà è `avvertimenti`, e il suo valore contiene il numero di volte che Guglielmo ha provato ad andare verso nord senza aver salutato il palo: 0 inizialmente, 1 dopo il primo avvertimento, 2 dopo il secondo, 3 - oh, no! La pazienza dei soldati è venuta meno.

La proprietà `avvertimenti` non è una proprietà della libreria standard, ma una che noi abbiamo creato per andare incontro ad un bisogno specifico esattamente come avevamo fatto per la proprietà `salutato`.

La nostra prima istruzione è `self.avvertimenti = 0`, che azzera il valore della proprietà `avvertimenti` dell'oggetto corrente - la stanza `centro_piazza`. La seconda istruzione è `palo.salutato = false`, che significa che il palo non è stato salutato. Ecco qua: la memoria del soldato è cancellata, e le azioni di Guglielmo sono dimenticate.

Caso 2: Andare a nord dopo aver prestato omaggio al palo

Guglielmo *va verso nord... ed ha salutato il palo*; un altro caso facile:

```
avvertimenti 0,       ! per contare gli avvertimenti dei soldati
    before [;
        Go:
            if (noun == s_obj) {
                self.avvertimenti = 0;
                palo.salutato = false;
            }
            if (noun == n_obj) {
                if (palo.salutato == true) {
                    print "^~Arrivederci, e buona giornata.~^";
                    return false;
                }
                else
                  { andando verso nord, senza aver salutato il palo }
            }
        ];
```

Tutto quello che dobbiamo fare è stampare un saluto sarcastico da parte dei soldati, e poi restituire `false`. Vi ricorderete che così facendo diciamo all'interprete di continuare a gestire l'azione, che in questo caso è un tentativo di andare a nord. Siccome questo è un movimento permesso, Guglielmo si ritroverà nella parte nord della piazza del villaggio, che definiremo tra poco.

Caso 3: Andare a nord senza aver reso omaggio al palo

Questo ci lascia soltanto con l'ultimo caso: *andando verso nord, senza aver salutato il palo*.

Questo caso è più complesso degli altri, visto che dobbiamo codificare il meccanismo "tre strike e sei fuori". Abbozziamolo un po' meglio:

```
avvertimenti 0,       ! per contare gli avvertimenti dei soldati
    before [;
        Go:
            if (noun == s_obj) {
                self.avvertimenti = 0;
                palo.salutato = false;
            }
            if (noun == n_obj) {
                if (palo.salutato == true) {
                    print "^~Arrivederci, e buona giornata.~^";
                    return false;
                }   ! fine (palo salutato)
                else {
```

```
                    self.avvertimenti = self.avvertimenti + 1;
                    switch (self.avvertimenti) {
                        1:  primo tentativo di andare a nord
                        2:  secondo tentativo di andare a nord
                        default: terzo e ultimo tentativo di andare a
                                                                nord
                    }
                }
            }
        ];
```

Anzitutto dobbiamo contare quante volte Guglielmo ha provato ad andare a nord. `avvertimenti` è la variabile che contiene questo numero, quindi noi aggiungiamo uno a qualunque valore essa contenga: `self.avvertimenti = self.avvertimenti + 1`. Poi, a seconda del valore di questa variabile, dobbiamo decidere quale azione intraprendere: primo tentativo, secondo tentativo o confronto finale. Potremmo usare tre diverse istruzioni if:

```
if (self.avvertimenti == 1) { primo tentativo di andare a nord }
if (self.avvertimenti == 2) { secondo tentativo di andare a nord }
if (self.avvertimenti == 3) { ultimo tentativo di andare a nord }
```

o un paio di `if` annidati:

```
if (self.avvertimenti == 1) { primo tentativo di andare a nord }
else {
    if (self.avvertimenti == 2) {secondo tentativo di andare a nord}
    else { ultimo tentativo di andare a nord }
}
```

ma, per una serie di controlli sulla stessa variabile, un'istruzione `switch` è generalmente il modo più pulito per ottenere lo stesso risultato.

La sintassi generica per un'istruzione `switch` è:

```
switch(espressione) {
    valore1: cosa succede quando l'espressione vale valore1
    valore2: cosa succede quando l'espressione vale valore2
    ....
    valoreN: cosa succede quando l'espressione vale valoreN
    default: cosa succede quando l'espr. ha qualsiasi altro valore
}
```

Questo significa che, a seconda del valore corrente dell'espressione, noi possiamo ottenere diversi esiti. Ricordate che l'`espressione` può essere una variabile `Globale` o locale, una proprietà di un oggetto, una delle variabili definite nella libreria o qualsiasi altra espressione che possa avere più di un valore. Potete scrivere `switch (x)`, se `x` è una variabile definita, ma anche, ad esempio, `switch (x+y)`, se sia `x` che `y` sono variabili definite. Quei *cosa succede quando...* sono collezioni d'istruzioni che implementano l'effetto desiderato per un certo valore dell'espressione valutata.

Anche se l'istruzione `switch (espressione) { ... }` ha bisogno di un paio di graffe, non ce n'è invece bisogno attorno ad ogni singolo "caso", non importa da quante istruzioni è composto. Come immaginiamo, i casi 1 e 2 contengono soltanto una singola istruzione `print_ret` ciascuno, così ci muoveremo

velocemente oltre, al ben più interessante terzo caso - quando `self.avvertimenti` vale 3. Avremmo potuto scrivere questo:

```
switch (self.avvertimenti) {
    1: primo tentativo di andare a nord
    2: secondo tentativo di andare a nord
    3: ultimo tentativo di andare a nord
}
```

ma usare la parola `default` – che significa "ogni valore non ancora interessato" – è una pratica di sviluppo migliore; è meno facile produrre risultati fuorvianti se per qualche ragione non prevista il valore di `self.avvertimenti` non è 1, 2 o 3 come previsto. Ecco il resto del codice (con un po' di testo omesso):

```
self.avvertimenti = self.avvertimenti + 1;
switch (self.avvertimenti) {
    1:  print_ret "...";
    2:  print_ret "...";
    default:
        print "^~Va bene, ";
        style underline; print "Herr"; style roman;
        print  " Tell, ora siete nei guai. Ve l'ho chiesto
                ...
                tiglio che cresce nella piazza del mercato.^";
        move mela to figlio;
        PlayerTo(mercato);
        return true;
}
```

La prima parte in realtà visualizza solo un mucchio di testo, ed è un po' più complicata del solito perché abbiamo voluto aggiungere enfasi alla parola "Herr" usando la sottolineatura (che in realtà appare come *corsivo* sulla maggior parte degli interpreti). Poi ci assicuriamo che Walter abbia la mela (nel caso non gliela avessimo data prima, durante il gioco), ci spostiamo nella stanza finale usando `PlayerTo(mercato)` e finalmente restituiamo `true` per comunicare all'interprete che abbiamo gestito da soli questa parte dell'azione `Go`.

E così, alla fine, ecco il codice completo per il centro della piazza, l'oggetto più complicato di tutto il gioco:

```
[TYPE]
    Room centro_piazza "In mezzo alla piazza"
        with
            description
                "C'è meno folla al centro della piazza; la maggior parte
                 della gente preferisce tenersi il più lontano possibile
                 dal palo che troneggia in questo luogo, reggendo
                 quell'assurdo cappello cerimoniale. Un gruppo di soldati
                 rimane nei pressi, osservando chiunque passi di qui.",
            n_to piazza_nord,
            s_to piazza_sud,
            avvertimenti 0, !per contare gli avvertimenti dei soldati
            before [;
                Go:
                    if (noun == s_obj) {
                            self.avvertimenti = 0;
                            palo.salutato = false;
                    }
                    if (noun == n_obj) {
```

```
                if (palo.salutato == true) {
                      print "^~Arrivederci, e buona giornata.~^";
                      return false;
                }   ! fine (palo salutato)
                else {
                    self.avvertimenti = self.avvertimenti + 1;
                        switch (self.avvertimenti) {
                            1:  print_ret
                                   "Un soldato ti sbarra la
                                    strada.     ^^     ~Hey,     tu,
                                    spilungone;  hai dimenticato le
                                    buone maniere? Forse sarebbe il
                                    caso di fare un bel saluto al
                                    cappello   del   balivo,   non
                                    trovi?~";
                            2:  print_ret
                                   "^~Ti conosco, Tell, sei uno che
                                    porta  solo  guai,  vero?  Non
                                    vogliamo   teste   calde   qui,
                                    quindi fai il bravo ragazzo e
                                    porgi il tuo saluto al dannato
                                    cappello. Fallo ora, non voglio
                                    chiedertelo di nuovo...~";
                            default:
                                   print "^~Va bene, ";
                                   style underline; print "Herr";
                                   style roman;
                                   print
                                       " Tell, ora siete nei guai.
                                        Ve l'ho chiesto
                                        gentilmente,  ma  voi  siete
                                        troppo     orgoglioso     e
                                        troppo  stupido.  Penso  che
                                        il balivo voglia scambiare
                                        qualche parola con voi.~ ^^
                                        Dicendo   ciò,   i   soldati
                                        prendono  te  e  Walter  e,
                                        mentre  il  sergente  corre
                                        via  a  cercare  Gessler,  il
                                        resto   degli   uomini   vi
                                        trascina rudemente verso il
                                        vecchio  tiglio  che  cresce
                                        nella       piazza       del
                                        mercato.^";
                                   move mela to figlio;
                                   PlayerTo(mercato);
                                   return true;
                        }   ! fine switch
                }   ! fine (palo non salutato)
    }   ! fine (noun == n_obj)
];
```

La parte nord della piazza

L'unico modo per arrivare qua è quello di salutare il palo e poi andare a nord; non molto probabile, ma il buon design di un gioco consiste nel predire l'imprevedibile.

```
[TYPE]
    Room piazza_nord "Lato nord della piazza"
        with   description
                   "Una piccola strada conduce verso nord, lasciando la
                    piazza affollata. Al centro della piazza, di poco
                    più a sud, vedi ancora il palo e il cappello.",
               n_to [;
                   deadflag = 3;
                   print_ret  "Con Walter al tuo fianco, lasci la piazza
                               percorrendo la strada verso nord, andando
                               verso la conceria di Johansson.";
               ],
               s_to "Non vuoi assolutamente passarci di nuovo.";
```

C'è solo una nuova caratteristica in questa stanza: il valore della proprietà `n_to` è una routine, che l'interprete esegue quando Guglielmo prova a lasciare la piazza dal lato nord. Tutto quello che fa la routine è assegnare il valore 3 alla variabile di libreria `deadflag`, stampare un messaggio di conferma e restituire `true`, concludendo così l'azione.

A questo punto l'interprete noterà che `deadflag` non è più uguale a zero e terminerà il gioco. In effetti l'interprete controlla `deadflag` alla fine di *ogni* turno; questi sono i valori che si aspetta di trovare:

- 0 : questo è lo stato normale; il gioco continua
- 1 : il gioco è finito. L'interprete visualizza "Sei morto".
- 2 : il gioco è finito. L'interprete visualizza "Hai vinto".
- ogni altro valore : il gioco è finito, ma non ci sono nella libreria messaggi appropriati. Piuttosto l'interprete cerca una routine **entry point** chiamata `DeathMessage` - che noi dobbiamo fornire - dove noi possiamo definire i nostri "messaggi di conclusione" personalizzati.

In questo gioco non impostiamo mai `deadflag` a 1, ma usiamo i valori 2 e 3. Così dovremmo definire una routine `DeathMessage` per dire al giocatore ciò che ha fatto:

```
[TYPE]
    [ DeathMessage; print "Hai rovinato una bellissima leggenda."; ];
```

Il nostro gioco ha solo un finale personalizzato, così la semplice routine `DeathMessage` che abbiamo scritto è sufficiente per i nostri scopi. Se volete preparare finali multipli per un gioco, potete specificare i messaggi adatti controllando il valore corrente della variabile `deadflag`:

```
    [ DeathMessage;
       if (deadflag == 3) print "Lasci Rossella O'Hara al suo destino";
       if (deadflag == 4) print "Stringi Rossella in un abbraccio
                                 appassionato";
```

```
        if (deadflag == 5) print "Sei riuscito a divorziare da
                                  Rossella";
        ...
    ];
```

Ovviamente, dovete assegnare il valore appropriato a `deadflag` quando il gioco raggiunge uno di questi finali.

Abbiamo quasi finito. Nel capitolo conclusivo di questo gioco parleremo del tiro decisivo di Guglielmo con l'arco.

9. Gugliemo Tell: la fine è vicina

Rimangono da definire ancora un po' di oggetti, così parleremo di questi per primi. Poi spiegheremo come fare aggiunte al repertorio di verbi standard di Inform, e come definire le azioni che vengono invocate da questi verbi.

La piazza del mercato

La stanza "piazza del mercato" è irrilevante, e l'`albero` che vi cresce ha una sola caratteristica interessante:

```
[TYPE]
    Room mercato "Piazza del mercato"
        with   description
                   "Il mercato di Altdorf, vicino alla piazza
                    principale, è stato velocemente sgombrato dai
                    banchi. Un gruppo di soldati ha spinto indietro la
                    folla per lasciare spazio libero di fronte al tiglio
                    che è cresciuto qui fin da quando la gente ha
                    memoria. Normalmente esso fa ombra ai vecchi della
                    città che si ritrovano qui per chiacchierare,
                    guardare le ragazze e giocare a carte. Oggi però,
                    esso è solitario... eccetto che per Walter, che è
                    stato legato al tronco. A circa quaranta metri
                    dall'albero, tu sei tenuto bloccato da due degli
                    uomini del balivo.",
               cant_go  "Cosa?  E  lasceresti  tuo  figlio  legato  a
               quell'albero?";

    Object  albero "tiglio" mercato
        with   name 'albero' 'tiglio' 'tronco',
               description "Un grosso albero.",
               before [;
                   FireAt:
                       if (NotBowOrArrow(second)) return true;
                       deadflag = 3;
                       print_ret "La mano ti trema, e la freccia vola
                                  alta,  andando  a  colpire  il  tronco,
                                  almeno  una  spanna  sopra  la  testa  di
                                  Walter.";
               ],
        has scenery;
```

La proprietà `before` dell'albero intercetta l'azione `FireAt`, che definiremo tra poco. Quest'azione è il risultato di un comando come SPARA ALL'ALBERO CON L'ARCO - potremmo simularla con il comando `<<FireAt albero arco>>` - ed ha bisogno di una cura particolare per assicuarsi che il secondo oggetto sia un'arma adatta. Per gestire comandi stupidi come SPARA ALL'ALBERO CON HELGA dobbiamo verificare che il secondo oggetto sia l'arco, una delle frecce, o niente (accettiamo anche SPARA ALL'ALBERO).

Siccome questo è un controllo abbastanza complesso che dobbiamo eseguire in diversi punti, è appropriato scrivere una routine dedicata a svolgere tale compito.

Una digressione: lavorare con le routine

Una **routine indipendente** (standalone routine), allo stesso modo delle familiari routine incorporate come valore di una proprietà come `before` o `each_turn`, è semplicemente una collezione d'istruzioni da eseguire. Le principali differenze sono nel contenuto, nel momento in cui vengono invocate e nel valore che restituiscono di default:

- Mentre una routine incorporata deve contenere istruzioni che fanno qualcosa di appropriato per la proprietà a cui è associata, una routine indipendente può contenere istruzioni che fanno qualsiasi cosa voi vogliate. Avete la completa libertà riguardo a quello che fa la routine ed al valore che essa restituisce.
- Una routine incorporata è chiamata quando l'interprete sta trattando quella proprietà per quell'oggetto; voi fornite la routine, ma non controllate direttamente quando questa viene chiamata. Una routine indipendente, invece, è completamente sotto il vostro controllo; viene eseguita solo quando voi la chiamate esplicitamente.
- Se una routine incorporata esegue tutte le istruzioni che contiene e raggiunge la chiusura `];` senza incontrare nessuna forma d'istruzione `return`, allora restituisce il valore `False`. Nelle medesime circostanze, una routine indipendente restituisce il valore `True`. Esiste una buona ragione per questa differenza – normalmente risulta essere il comportamento naturale di default – ma talvolta può confondere i nuovi arrivati. Per evitare di generare confusione, espliciteremo sempre l'istruzione `return` nelle nostre routine.

Quello che generalmente viene fuori da tutto ciò è questo: *se* avete una collezione di istruzioni che eseguono un qualche compito specifico, *e* avete bisogno di eseguire queste stesse istruzioni in più di un punto del vostro gioco, *allora* spesso è più sensato trasformare queste istruzioni in una routine indipendente. I vantaggi sono: scrivete le istruzioni una volta sola, e così ogni modifica successiva alla routine sarà più facile da realizzare; inoltre il gioco diventa più semplice e più facile da leggere. Vediamo qualche semplice esempio; prendiamo come esempio questi due cibi poco invitanti:

```
Object "panino raffermo al prosciutto"
    with   name 'panino' 'raffermo' 'prosciutto',
           description "Non sembra molto appetitoso.",
...

Object "vecchia crostata alla marmellata"
    with   name 'vecchia' 'crostata' 'marmellata' 'dolce',
           description " Non sembra molto appetitoso.",
...
```

Le due descrizioni sono identiche: forse potremmo visualizzarle usando una routine?

```
[ Immangiabile; print_ret " Non sembra molto appetitoso."; ];

Object "panino raffermo al prosciutto"
    with   name 'panino' 'raffermo' 'prosciutto',
           description [; Immangiabile(); ],
...

Object "vecchia crostata alla marmellata"
    with   name 'vecchia' 'crostata' 'marmellata' 'dolce',
           description [; Immangiabile(); ],
...
```

Questo non è un approccio molto realistico – ci sono modi più eleganti per evitare di scrivere la stessa stringa due volte – ma funziona, e mostra come possiamo definire una routine che fa qualcosa di utile e poi chiamarla dove ne abbiamo bisogno.

Ecco un altro semplice esempio che mostra come, restituendo un valore, la routine può riferire un risultato al pezzo di codice che l'ha chiamata. Abbiamo usato un paio di volte il test `if (self has visited) ...;` potremmo creare una routine che esegue lo stesso controllo e restituisce vero o falso per indicare cosa ha scoperto:

```
[ gia_stato_qui;
    if (self has visited) return true;
    else                  return false;
];
```

E poi riscriviamo i nostri controlli come `if (gia_stato_qui() ==true) ...;` non è più breve o più veloce, ma forse è più descrittivo riguardo a cosa sta accadendo.

Un ultimo esempio sull'uso delle routine. Così come abbiamo controllato `if (self has visited) ...`, abbiamo anche controllato qualche `volta if (location has visited) ...`, così *potremmo* scrivere un'altra routine per fare il controllo.

```
[ gia_stato_stanza;
    if (location has visited) return true;
    else                      return false;
];
```

Ma le due routine sono molto simili; la sola differenza è il nome della variabile - `self` o `location` - che viene controllata. Un approccio migliore potrebbe essere quella di rifare la nostra routine `gia_stato_qui` in modo che svolga entrambi i compiti, ma dobbiamo dirgli in qualche modo quale variabile deve controllare. Questo è facile: scriviamo la routine in modo che richieda un **argomento**:

```
[ gia_stato luogo;
    if (luogo has visited)   return true;
    else                     return false;
];
```

Notate che il nome dell'argomento l'abbiamo inventato in modo che descriva il proprio contenuto. La routine non si preoccupa del fatto che noi lo definiamo come "`x`", "`luogo`" o "`piero_piero`". Qualunque sia il suo nome, l'argomento

funziona come segnaposto per il suo valore (nel nostro esempio una delle due variabili, `self` o `location`), che noi dobbiamo fornire quando chiamiamo la routine:

```
if (gia_stato(self) == true) ...
if (gia_stato(location) == true) ...
```

Nella prima riga forniamo `self` come argomento per la routine. Comunque alla routine non interessa da dove viene l'argomento; lei vede solo un valore che conosce come `luogo`, e che usa per verificare l'attributo `visited`. Nella seconda riga forniamo `location` come argomento, ma la routine vede soltanto un altro valore nella sua variabile `luogo`. La variabile `luogo` è chiamata **variabile locale** della routine `gia_stato`, una variabile a cui deve essere assegnato un valore adatto ogni volta che la routine viene chiamata. In effetti alla routine non interessa da dove viene l'argomento, basta che esso rappresenti un oggetto stanza; potremmo anche controllare una stanza nominata esplicitamente:

```
if (gia_stato(centro_piazza) == true) ...
```

Ricordate che:

1. Tutte le routine terminano il loro compito prima o poi, sia a causa di una esplicita istruzione `return`, `rtrue` o `rfalse`, sia a causa del raggiungimento del simbolo di chiusura `]` posto alla fine della routine.

2. Tutte le routine restituiscono (`return`) un valore, che può essere `true`, o `false`, o un qualsiasi altro numero. Questo valore è determinato dall'istruzione `return`, `rtrue` o `rfalse`, o dal simbolo di chiusura `]` posto al termine della routine (nel qual caso si applica la regola di default: le routine indipendenti restituiscono `true`, le routine incorporate restituiscono `false`). Abbiamo offerto questo esempio di una routine incorporata in "Aggiungiamo un po' di scenografia" al Capitolo 7. L'istruzione `return false` è ridondante: potremmo rimuoverla senza influenzare il comportamento della routine, dal momento che `]` agisce esattamente come `return false`:

   ```
   found_in [;
       if (location == strada or vicino_piazza or piazza_sud or
           centro_piazza or piazza_nord or mercato) return true;
       return false;
   ],
   ```

   D'altra parte, il fatto che una routine restituisca un certo valore non significa che sia *necessario* usare sempre tale valore: potete semplicemente ignorarlo se lo desiderate. La routine `Immangiabile` che abbiamo mostrato precedentemente in questo capitolo contiene una istruzione `print_ret` e quindi restituisce sempre il valore `true`, ma non vi abbiamo prestato alcuna attenzione; il solo obiettivo della routine era visualizzare del testo. Confrontate tale routine con la routine `gia_stato`, che *non fa altro* che restituire `true`; se lo avessimo ignorato, allora la chiamata alla routine sarebbe stata solo una perdita di tempo.

Per alcune routine incorporate, il valore restituito dalla routine è importante; per altre no. Abbiamo visto finora le seguenti proprietà il cui valore può essere una routine incorporata:

Il valore di ritorno è importante	Il valore di ritorno non importa
`after [; ... ],`	`cant_go [; ... ],`
`before [; ... ],`	`description [; ... ],`
`found_in [; ... ],`	`each_turn [; ... ],`
`n_to [; ... ],` et similia.	`initial [; ... ],`

Per i dettagli completi su quale proprietà della libreria può essere una routine incorporata, e quale restituisce valori significativi, si consultino "Le proprietà degli oggetti" all'Appendice F e l'Appendice §A2 dell'*Inform Designer's Manual*.

Torniamo al mercato

Dopo tutta questa introduzione, torniamo finalmente all'azione `FireAt`. Nel nostro esempio dobbiamo controllare le caratteristiche di un oggetto e dopo visualizzare possibilmente un messaggio. Non sappiamo esattamente *quale* oggetto deve essere controllato, così dobbiamo scrivere la nostra routine in modo generalizzato, facendo sì che sia capace di controllare *qualsiasi* oggetto vogliamo; ovvero dobbiamo fornire l'oggetto da controllare come argomento. Ecco la routine:

```
[TYPE]
    [ BowOrArrow o;
        if (o == arco or nothing || o ofclass Arrow) return true;
        print "Non è granché come arma, vero?";
        return false;
    ];
```

La routine è stata pensata per verificare qualsiasi oggetto che le sia stato passato come argomento `o`; questo vuol dire che possiamo chiamare la routine in questo modo:

```
BowOrArrow(proprietaria)
BowOrArrow(mela)
BowOrArrow(albero)
BowOrArrow(arco)
```

Se viene fornito un oggetto come Helga, la `mela` o l'`albero`, la routine stamperà un messaggio e restituirà `false` per indicare che in effetti questo oggetto non è un'arma adatta. Dato, invece, l'oggetto `arco`, od ogni oggetto definito nella classe `Arrow`, restituirà silenziosamente `true`, per affermare che l'oggetto è un'arma. Il controllo che facciamo è:

```
if (o == arco or nothing || o ofclass Arrow) ...
```

che è soltanto un modo più breve di dire:

```
if (o == arco || o == nothing || o ofclass Arrow) ...
```

Il risultato è che noi chiediamo tre cose: `o` è l'oggetto `arco`? Oppure è `nothing`? Oppure è un qualunque oggetto membro della classe `Arrow`?

Questo significa che il valore restituito quando chiamiamo `BowOrArrow(albero)` è `false`, mentre il valore restituito quando chiamiamo `BowOrArrow(arco)` è `true`. In termini più generali, il valore restituito quando chiamiamo `BowOrArrow(second)` sarà o `true` o `false`, a seconda delle caratteristiche dell'oggetto definito dal valore della variabile `second`. Così possiamo scrivere questa coppia d'istruzioni nella proprietà `before` di un oggetto:

```
if (BowOrArrow(second) == true) {
    Questo oggetto ha a che fare con un freccia lanciatagli contro.
}
return true;
```

e l'effetto potrebbe essere:

- `second` *non è* un'arma: `BowOrArrow` visualizza un messaggio "non essere stupido" e restituisce il valore `false`, l'istruzione `if` reagisce a quel valore e ignora le istruzioni seguenti; oppure
- `second` è un'arma: `BowOrArrow` non visualizza nulla e restituisce il valore `true`, l'istruzione `if` reagisce a quel valore eseguendo le istruzioni successive per produrre un comportamento adeguato all'essere colpito da una freccia; e quindi
- in entrambi i casi, l'istruzione `return true` termina l'intercettazione dell'azione `FireAt` dell'oggetto.

L'intera routine `BowOrArrow()` è un po' complessa, ma il resto dell'azione `FireAt` è lineare. Una volta che l'albero ha determinato che è stato colpito con qualcosa di adatto, assegna a `deadflag` il valore 3 - la fine "Hai rovinato..." - stampa un messaggio, ed ha finito.

Gessler il governatore

Non c'è niente nella definizione di Gessler che non abbiamo già incontrato in precedenza:

```
[TYPE]
    NPC     balivo "balivo" mercato
        with
            name 'governatore' 'balivo' 'hermann' 'gessler',
            description
                "Basso e robusto, ma dal volto subdolo e affilato,
                 Gessler usa il potere affidatogli opprimendo la comunità
                 locale.",
            initial [;
                print  "Gessler     sta     osservando     da     lontano,
                        con un sogghigno sul volto.^";
                if (location hasnt visited)
                    print  "~A quanto pare hai bisogno di una buona
                            lezione, stupido. Nessuno deve attraversare
                            la piazza senza prestare omaggio a Sua
                            Altezza Imperiale Alberto; nessuno, hai
```

```
                    capito? Potrei farti decapitare per
                    tradimento, ma voglio essere clemente. Se ti
                    comporterai ancora follemente, non potrai
                    aspettarti alcuna pietà da parte mia, ma
                    questa volta sei libero di andare... non
                    appena mi avrai dimostrato la tua abilità di
                    arciere colpendo questa mela da dove ti
                    trovi. Non dovrebbe essere troppo difficile.
                    Sergente, prenda; la appoggi sulla testa di
                    quel piccolo bastardo.~^";
        ],
        life [;
               Talk:
                   print_ret  "Non hai intenzione di rivolgergli la
                               parola.";
        ],
        before [;
               FireAt:
                   if (BowOrArrow(second) == true) {
                       deadflag = 3;
                       print_ret  "Prima che i soldati possano
                                   reagire, ti volti e scagli una
                                   freccia contro Gessler; la tua
                                   frecca colpisce il suo cuore ed
                                   egli muore miseramente. La folla
                                   ha un sussulto, seguito da un
                                   applauso.";
                   }
                   return true;
        ],
    has male;
```

Come la maggior parte degli NPC, Gessler ha una proprietà `life` che si occupa delle azioni che si applicano solo agli oggetti animati. Questa risponde soltanto a `Talk` (ovvero al comando PARLA AL GOVERNATORE).

L'attributo `male` è ridondante, visto che in italiano non esiste il genere neutro e che quindi gli oggetti (animati e non) sono di default di genere maschile a meno che non venga specificato altrimenti (NdT).

Walter e la mela

Visto che è stato con voi tutto il tempo, è il momento di definire Walter:

```
[TYPE]
    NPC figlio "tuo figlio"
        with
            name 'figlio' 'tuo' 'ragazzo' 'bambino' 'walter',
            description [;
                if (location == mercato)
                    print_ret "Ti sta guardando, cercando di apparire
                        coraggioso e di rimanere fermo. Le sue braccia
                        sono legate dietro al tronco, e la mela è stata
                        posata tra i suoi capelli biondi.";
                else
                    print_ret "Un tranquillo ragazzo biondo di otto
                        primavere, rapido ad imparare lo stile di vita dei
                        montanari.";
            ],
            life [;
                Give:
                    score = score + 1;
                    move noun to self;
                    print_ret "~Grazie, Papi.~";
                Talk:
                    if (location == mercato)
                        print_ret "~Stai calmo, figliolo, e confida in
                            Dio.~";
                    else
                        print_ret "Spieghi a tuo figlio la tua visione
                            della vita.";
                ],
                before [;
                    Examine,Listen,Salute,Talk: return false;
                    FireAt:
                        if (location == mercato) {
                            if (BowOrArrow(second) == true) {
                            deadflag = 3;
                            print_ret "Oops! Sicuramente non volevi far
                                ciò...";
                            }
                            return true;
                        }
                        else return false;
                    default:
                        if (location == mercato)
                          print_ret "Le guardie non lo permetterebbero.";
                        else return false;
                ],
                found_in [; return true; ],
        has male proper scenery transparent;
```

I suoi attributi sono `male`[9] (è vostro figlio, dopo tutto), `proper` (così l'interprete non scriverà "il tuo figlio"), `scenery` (così non verrà elencato nella descrizione di ogni stanza) e `transparent` (poiché potete vedergli attraverso). No, questo è un

[9] Nella programmazione in Inform in italiano tale attributo è ridondante. Non essendovi il genere neutro nella nostra lingua INFIT di default è impostata perchè tutti gli ogetti siano considerati maschili se non diversamente specificato.

errore: un oggetto `transparent` non è fatto di vetro; è un oggetto che vi permette di vedere gli oggetti che possiede. Abbiamo fatto ciò perché vorremmo essere sempre in grado di ESAMINARE LA MELA anche se è Walter che la sta portando. Senza l'attributo `transparent`, sarebbe come se la mela fosse nella sua tasca o comunque non in vista; l'interprete risponderebbe "Non vedi nulla del genere".

Walter ha una properità `found_in` che automaticamente lo muove ad ogni turno nella stessa locazione del giocatore. Possiamo accettare tale impostazione dal momento che in un gioco così piccolo e semplice, egli in effetti vi segue dovunque. In mondi simulati più realistici, gli NPC di solito si muovono indipendentemente, ma noi non abbiamo bisogno di tanta complessità in questo caso.

Diverse proprietà di Walter controllano se `(location == mercato)`; ovvero, se il giocatore (e quindi Walter) si trova al momento in quella stanza. Gli eventi nella piazza del mercato sono tali che risposte specializzate sono più appropriate delle risposte generiche.

La proprietà `life` di Walter reagisce a `Give` (ovvero al comando DAI LA MELA A WALTER) e `Talk` (ovvero PARLA A TUO FIGLIO); durante `Give`, aumentiamo il valore della variabile di libreria `score`, ricompensando così la natura generosa del giocatore. La sua proprietà `before` è un po' più complessa. Essa dice:

1. Le azioni `Examine`, `Listen`, `Salute` e `Talk` sono sempre disponibili (quindi l'azione `Talk` viene passata alla proprietà `life` di Walter).
2. L'azione `FireAt` è permessa nella piazza del mercato, anche se con risultati sfortunati. In qualunque altro posto, essa attiva la risposta generica per `FireAt`, "Sembra pericoloso, non trovi?".
3. Tutte le altre azioni sono evitate nella piazza del mercato, mentre in altri luoghi permettiamo che seguano il loro normale svolgimento (grazie all'istruzione `return false`).

Il momento di gloria della mela è arrivato! La sua proprietà `before` risponde all'azione `FireAt` impostando la variabile `deadflag` a 2. Quando ciò succede, il gioco è finito; il giocatore ha vinto.

```
[TYPE]
    Object  mela "mela"
        with
            name 'mela',
            description [;
                if (location == mercato)
                    print_ret "A questa distanza puoi appena vederla.";
                else
                    print_ret "La mela è a chiazze verdi e marroni.";
            ],
            before [;
                Drop:
```

```
            print_ret    "Una    mela    vale    sempre    qualcosa,
            meglio tenersela.";
        Eat: print_ret "Helga te l'ha data per Walter...";
        FireAt:
            if (location ~= mercato) {
                if (BowOrArrow(second) == true) {
                    score = score + 1;
                    deadflag = 2;
                    print_ret "Con calma e fermezza incocchi una
                        freccia nell'arco, tendi la corda e prendi
                        la mira con più cura di quanto tu abbia mai
                        fatto in vita tua. Trattenendo il fiato,
                        senza un batter d'occhio, timorosamente,
                        rilasci la freccia. Questa vola attraverso
                        la piazza, verso tuo figlio, e pianta la
                        mela contro il tronco dell'albero. La folla
                        esulta; Gessler sembra decisamente
                        deluso.";
                }
                return true;
            }
            else return false;
    ];
```

E con ciò, abbiamo definito tutti gli oggetti. Così facendo abbiamo aggiunto un bel carico di nuovi nomi e aggettivi al dizionario del gioco, ma nessun verbo. Questo è il nostro ultimo compito.

Verbi, Verbi, Verbi

La libreria di Inform fornisce un insieme standad di circa un centinaio di azioni che il giocatore può eseguire; all'incirca una ventina di queste sono "meta-azioni" (come SALVA e QUIT) dirette all'interprete stesso, mentre le restanti operano all'interno del mondo virtuale. Avere un insieme di partenza così grande è sicuramente una benedizione; significa che molte delle azioni che un giocatore potrebbe provare sono già gestite, o dall'interprete che fa qualcosa di utile o spiegando perché non può farlo.

Nonostante ciò, la maggior parte dei giochi ha bisogno di definire azioni addizionali, e "Guglielmo Tell" non fa eccezione. Aggiungeremo quattro azioni da noi definite: `Untie` (slega), `Salute` (saluta, riverisci), `FireAt` (spara a) e `Talk` (parla).

Untie

Non è l'azione più utile, ma è la più semplice. Nella piazza del mercato, quando Walter è legato all'albero, è possibile che il giocatore sia tentato dall'idea di provare SLEGA WALTER o LIBERA WALTER; improbabile ma, come abbiamo detto prima, prevedere l'improbabile fa parte della creazione di Avventure Testuali. Per questa, e per tutte le nuove azioni, abbiamo bisogno di

due cose. Abbiamo bisogno di una definizione grammaticale, che specifichi la struttura delle frasi in Italiano che siamo preparati ad accettare:

```
[TYPE]
    Verb 'slega' 'sciogli' 'libera' 'rilascia'
        * noun                      -> Untie;
```

e abbiamo bisogno di una routine per gestire l'azione nella situazione standard (quando l'azione *non è* intercettata dalla proprietà `before` di un oggetto):

```
[TYPE]

    [ UntieSub; print_ret "Non dovresti provarci."; ];
```

La grammatica è meno complessa di come appare ad un primo colpo d'occhio:

1. I verbi italiani SLEGA, SCIOGLI, LIBERA e RILASCIA sono sinonimi.
2. L'asterisco `*` indica l'inizio di uno schema che definisce quali parole possono seguire il verbo.
3. In questo esempio c'è un solo schema: l'elemento "`noun`" rappresenta il nome di un oggetto che è correntemente "in scope" - presente nella stessa stanza del giocatore.
4. Il segno `->` indica l'azione che deve essere attivata.
5. Se il giocatore scrive qualcosa che corrisponde allo schema - uno di quei quattro verbi seguito dal nome di un oggetto presente nella stanza - l'interprete attiva un'azione `Untie`, che normalmente è gestita da una routine che abbia lo stesso nome dell'azione con il suffisso `Sub`. In questo esempio è la routine `UntieSub`.
6. La grammatica è disposta in questo modo solo per renderla più semplice da leggere. Tutti quegli spazi non sono importanti; avremmo potuto scrivere:

   ```
   Verb 'slega' 'sciogli' 'libera' 'rilascia' * noun -> Untie;
   ```

Mostriamo come tutto questo funziona nella strada di Altdorf:

```
Una strada di Altdorf
La piccola strada conduce verso nord alla piazza principale. La
gente del luogo sta sciamando nella città attraverso la porta a
sud, salutando a voce alta, offrendo prodotti in vendita,
scambiando notizie, informandosi con incredulità esagerata sui
prezzi delle merci esposte dai mercanti, i cui banchi rendono
ancora più difficile l'avanzare in mezzo alla folla.

"Stammi vicino, figliolo," dici, "altrimenti potresti perderti fra
tutta questa gente."

>SLEGA
Cosa vuoi slegare?

>SCIOGLI IL CANE
Non vedi nulla del genere.

>RILASCIA LA GENTE
Non hai bisogno di preoccuparti della gente.

>LIBERA TUO FIGLIO
Non dovresti provarci.
```

Questo esempio mostra quattro tentativi di usare la nuova azione. Nel primo il giocatore non menziona l'oggetto a cui si riferisce l'azione; l'interprete sa (da quel `noun` nella grammatica che implica che l'azione ha bisogno di un complemento oggetto) che manca qualcosa, e fornisce una risposta utile. Nel secondo caso, il giocatore menziona un oggetto che non è presente, che non è "in scope" (in effetti non c'è nessun cane in nessuna parte del gioco, ma l'interprete non vuole suggerire così *tanto* al giocatore). Nel terzo caso, l'oggetto è "in scope", ma la sua proprietà `before` intercetta l'azione `Untie` (e in effetti, visto che questo oggetto è della classe `Prop`, *tutte* le azioni eccetto `Examine`) per visualizzare un messaggio di diniego personalizzato. Finalmente, il quarto utilizzo si riferisce ad un oggetto che *non* intercetta l'azione, così l'interprete chiama la routine che gestisce di default l'azione - `UntieSub` - che visualizza un rifiuto generale ad eseguire l'azione.

I principi presentati qui sono quelli che dovreste generalmente usare: scrivere una routine per la gestione generica dell'azione che rifiuta di fare qualsiasi cosa (si veda, per esempio, SCHIACCIA o COLPISCI), o che esegue l'azione senza modificare lo stato del mondo (vedi, per esempio, SALTA o CANTA); poi intercettate quella non-azione (generalmente usando una proprietà `before`) per quegli oggetti che potrebbero essere un bersaglio legittimo per l'azione, e quindi fornire una risposta più specifica, eseguendo o rifiutando l'azione.

Nel caso di `Untie`, non ci sono oggetti nel gioco che possono essere slegati, così noi generiamo sempre un rifiuto di un qualche tipo.

Salute

La prossima azione è `Salute`, fornita nel caso Guglielmo scelga di prestare omaggio al cappello sul palo. Questa routine è il gestore generico dell'azione:

```
[TYPE]
    [ SaluteSub;
       if (noun has animate)
              print_ret (The) noun, " restituisce il saluto.";
       print_ret (The) noun, " non mostra alcuna reazione.";
    ];
```

Noterete che questa è leggermente più intelligente del gestore per `Untie`, visto che produce risposte differenti a seconda che l'oggetto salutato - contenuto nella variabile `noun` - sia animato o no. Ma fondamentalmente sta facendo lo stesso lavoro. Ed ecco la grammatica:

```
[TYPE]
    Verb 'inchinati' 'genuflettiti'
        *   'a'/'ad'/'all^'/'allo'/'alla'/'al'/'agli'/'ai'/
            'alle'/'verso' noun      -> Salute;

    Verb 'riverisci'
        * noun                       -> Salute;

    Verb 'presta' 'rendi'
        'omaggio'/'omaggi' 'a'/'ad'/'all^'/'allo'/'alla'/
        'al'/'agli'/'ai'/'alle'      -> Salute;
```

Questa grammatica dice che:

1. I verbi italiani `'inchinati'`, `'genuflettiti'`, `'riverisci'`, `'presta'` e `'rendi'` sono sinonimi.
2. I primi due (ma non il terzo) possono essere seguiti da una delle seguenti proposizioni A, AD, ALL', ALLO, ALLA, AL, AGLI, AI, ALLE o VERSO: le parole tra apici devono corrispondere esattamente, con la barra `/` che separa le varie alternative. Invece il quarto e il quinto possono essere seguiti da `'omaggio'` o `'omaggi'` e una delle proposizioni di cui sopra (eccetto VERSO).
3. Dopo di ciò viene il nome di un oggetto "in scope" - nella stessa stanza del giocatore.
4. Se il giocatore scrive qualcosa che corrisponde a questi schemi, l'interprete attiva un'azione `Salute`, che normalmente è gestita dalla routine `SaluteSub`.

Così noi possiamo scrivere INCHINATI AL CAPPELLO, GENUFLETTITI VERSO IL CAPPELLO, PRESTA OMAGGIO AL CAPPELLO, ma non semplicemente INCHINATI CAPPELLO o RENDI AL CAPPELLO. Inoltre possiamo scrivere RIVERISCI IL CAPPELLO, ma non RIVERISCI VERSO IL CAPPELLO. Non è perfetto, ma è un onesto tentativo per definire qualche nuovo verbo per gestire il saluto.

Ma supponiamo di pensare che ci sono altri modi in cui il giocatore può tentare ciò (ricordate, loro non sanno quali verbi abbiamo definito; stanno provando alla cieca, provando cose che secondo loro dovrebbero funzionare). Che ne pensate di SALUTA IL CAPPELLO, o OFFRI OMAGGIO AL CAPPELLO? Sembrerebbe abbastanza ragionevole, no? Peccato però che se noi avessimo scritto

```
Verb 'riverisci' 'saluta'
    * noun                      -> Salute;

Verb 'presta' 'rendi' 'dai' 'offri'
    *   'omaggio'/'omaggi' 'a'/'ad'/'all^'/'allo'/'alla'/
        'al'/'agli'/'ai'/'alle'  -> Salute;
```

avremmo provocato un errore di compilazione: *two different verb definitions refer to "saluta"*, due diverse definizioni di verbo per "saluta" (e lo stesso errore vale anche per offri e dai). `ItalianG.h`[10], il solo file di libreria di cui un principiante ha bisogno di studiare il contenuto, contiene queste righe:

```
Verb 'saluta'
    *   creature      -> WaveHands;

Verb 'dai' 'paga' 'offri' 'da'
    *   held 'a'/'ad'/'all^'/'allo'/'alla'/'al'/'agli'/'ai'/
        'alle' creature
                      -> Give
```

[10] grammar.h nella versione originale inglese. NdT.

```
        *   'a'/'ad'/'all^'/'allo'/'alla'/'al'/'agli'/'ai'/
            'alle' creature held
                        -> Give reverse;
```

Il problema è che i verbi `saluta`, `dai` e `offri` sono già definiti nella libreria, e un verbo può apparire in una sola definizione `Verb`. La soluzione sbagliata: modificare `ItalianG.h` per modificare *fisicamente* la definizione per `'saluta'` (non è quasi mai una buona idea apportare cambiamenti ai file della libreria standard o di `infit`). La soluzione giusta: usare `Extend` per aggiungere *logicamente* qualcosa alla definizione:

```
[TYPE]
    Extend 'saluta' first
        * noun          -> Salute;
    Extend 'dai'
        *   'omaggio'/'omaggi' 'a'/'ad'/'all^'/'allo'/
            'alla'/'al'/'agli'/'ai'/'alle'/'verso' noun
                        -> Salute;
```

e così l'effetto è come se avessimo editato `ItalianG.h` e lo avessimo modificato così:

```
    Verb 'saluta'
        * noun                              -> Salute
        * creature                          -> WaveHands;

    Verb 'dai' 'paga' 'offri' 'da'
    *   held 'a'/'ad'/'all^'/'allo'/'alla'/'al'/'agli'/'ai'/
        'alle' creature
                                            -> Give
    *   'a'/'ad'/'all^'/'allo'/'alla'/'al'/'agli'/'ai'/
        'alle' creature held
                                            -> Give reverse
    *   'omaggio'/'omaggi' 'a'/'ad'/'all^'/'allo'/
        'alla'/'al'/'agli'/'ai'/'alle'/'verso' noun
                                            -> Salute;
```

Notate come nella definizione di `Extend 'Saluta'` abbiamo aggiunto la parola chiave `first`. Essa mette la nuova definizione logica al primo posto nell'elenco di definizioni del verbo. L'elenco delle definizioni del verbo infatti ha importanza dal momento che il programma le scorrerà una ad una fino a trovare la prima che si può applicare. Pertanto sono a disposizione del programmatore tre parole chiave che possono essere associate ad `extend`. `first` che mette la nuova definizione al primo posto nell'elenco delle definizioni per quel verbo; `Last` che la mette in fondo (comportamento predefinito); e `replace` che sostituisce invece la nuova definizione a quella vecchia. NdT.

Ora il giocatore può SALUTARE qualsiasi oggetto, e DARE o (OFFRIRE) OMAGGIO a qualsiasi oggetto.

(Poiché DAI, PAGA, OFFRI e DA sono definiti come sinonimi, il giocatore può anche PAGARE OMAGGIO A qualcosa, ma è improbabile che qualcuno noti questa minima aberrazione; i giocatori in genere sono troppo occupati a provare possibilità logiche.)

FireAt

Al solito, mostriamo il gestore generico dell'azione:

```
[TYPE]
    [ FireAtSub;
       if (noun == nothing) print_ret "Cosa? Scagliare frecce a caso?";
       if (NotBowOrArrow(second)) return true;
       print_ret "Sembra pericoloso, non trovi?";
    ];
```

Seguito dalla grammatica:

```
[TYPE]
    Verb 'spara' 'mira' 'scaglia'
        *   noun                                -> FireAt
        *   'con' noun                          -> FireAt
        *                                       -> FireAt
        *   'a'/'ad'/'all^'/'allo'/'alla'/'al'/'agli'/'ai'/
            'alle'/'su'/'sul'/'sullo'/'sull^'/'sulla'/'sui'/
            'sugli'/'sulle'/'sopra'/'contro' noun
                                                -> FireAt
        *   'a'/'ad'/'all^'/'allo'/'alla'/'al'/'agli'/'ai'/
            'alle'/'su'/'sul'/'sullo'/'sull^'/'sulla'/'sui'/
            'sugli'/'sulle'/'sopra'/'contro' noun
            'con' noun
                                                -> FireAt
        *   'con' noun 'a'/'ad'/'all^'/'allo'/'alla'/'al'/'agli'/
            'ai'/'alle'/'su'/'sul'/'sullo'/'sull^'/'sulla'/
            'sui'/'sugli'/'sulle'/'sopra'/'contro' noun
                                                -> FireAt reverse
        *   noun 'a'/'ad'/'all^'/'allo'/'alla'/'al'/'agli'/'ai'/
            'alle'/'su'/'sul'/'sullo'/'sull^'/'sulla'/'sui'/
            'sugli'/'sulle'/'sopra'/'contro' noun
                                                -> FireAt reverse;
```

Questa è la grammatica più complessa che scriveremo, ed è la prima che offre diverse opzioni per le parole che seguono il verbo iniziale. La prima riga della grammatica:

```
    *                                   -> FireAt
```

ci permette di scrivere soltanto SPARA (o MIRA, o SCAGLIA). Le due righe che seguono:

```
    * noun                              -> FireAt

    * 'con' noun                        -> FireAt
```

supportano SPARA UNA FRECCIA e SPARA CON L'ARCO (e anche qualcosa di meno adatto, come SPARA L'ALBERO). Il gruppo di righe successivo

```
    *   'a'/'ad'/'all^'/'allo'/'alla'/'al'/'agli'/'ai'/
        'alle'/'su'/'sul'/'sullo'/'sull^'/'sulla'/'sui'/
        'sugli'/'sulle'/'sopra'/'contro' noun
                                        -> FireAt
```

accetta SPARA ALL'ALBERO, SPARA ALLA MELA, e così via. Notate che c'è soltanto un punto e virgola in tutta la grammatica, proprio alla fine.

Le prime due istruzioni in `FireAtSub` gestiscono la prima riga della grammatica: SPARA (o MIRA, o SCAGLIA) da solo. Se il giocatore scrive solo quello, sia `noun` che `second` non conterranno nulla, e così noi rifiutiamo il tentativo con il messaggio "Cosa?..". Altrimenti abbiamo per lo meno un valore nella variabile `noun`, e possibilmente un valore anche in `second`, così generalmente controlleremo se `second` è qualcosa che può sparare o essere sparato, e poi rifiuteremo l'azione con il messaggio "Sembra pericoloso".

Ci sono un paio di motivi per cui potreste trovare un po' complicata questa grammatica. Il primo è che in qualche riga la parola `noun` appare due volte: dovete ricordare che in questo contesto `noun` è un elemento di parsing (riconoscimento della frase) che controlla ogni singolo oggetto visibile da parte del giocatore. Perciò la riga (semplificato):

```
* 'a' noun 'con' noun -> FireAt
```

corrisponde a SPARA A *qualche_bersaglio_visibile* CON *`qualche_arma_visibile;`* forse creando confusione, ma il valore dell'oggetto bersaglio è poi memorizzato nella variabile `noun` e il valore dell'oggetto arma nella variabile `second`.

La seconda difficoltà può essere data dagli ultimi due blocchi di definizione della grammatica. Mentre nelle righe precedenti il primo oggetto individuava il bersaglio ed il secondo, se presente, l'arma, questi blocchi finali corrispondono a SPARA CON *`qualche_arma_visibile`* A *`qualche_bersaglio_visibile`* e SPARA *`qualche_arma_visibile`* A *`qualche_bersaglio_visibile`* - i due oggetti sono menzionati in ordine inverso. Se non facessimo nulla, la nostra `FireAtSub` sarebbe parecchio confusa a questo punto, ma noi possiamo scambiare di posto i due oggetti, rimettendoli nell'ordine giusto, aggiungendo quella parola `reverse` alla fine della riga, e allora `FireAtSub` funzionerà nello stesso modo in ogni caso.

A questo punto, in questa versione italiana, sarà il caso di estendere anche il verbo LANCIA (definito dalla libreria italiana) per gestire l'azione LANCIA LA FRECCIA ALLA MELA:

```
[TYPE]
    Extend only 'lancia' first
        *   noun 'a'/'ad'/'all^'/'allo'/'alla'/'al'/'agli'/'ai'/
            'alle'/'su'/'sul'/'sullo'/'sull^'/'sulla'/'sui'/
            'sugli'/'sulle'/'sopra'/'contro' noun
                                        -> FireAt reverse;
```

Notate qui l'inserimento tra `extend` e il verbo dell'ulteriore parola chiave `only`. Essa è utile per dire al programma di estendere la definizione solo del verbo LANCIA e non di tutti i suoi sinonimi contenuti nella libreria (lancia nella libreria è anche sinonimo di lascia ad esempio).

Prima di abbandonare l'azione FireAt, aggiungeremo un altro piccolo pezzo di grammatica:

```
[TYPE]
    Extend 'colpisci' replace
        *   noun                        -> FireAt;
```

L'uso della direttiva `Extend` che già conosciamo, questa volta con l'aggiunta della parola chiave `replace` (che abbiamo già menzionato). L'effetto è quello di sostituire la nuova grammatica qui definita con quella contenuta in `ItalianG.h`, così che COLPISCI, UCCIDI, AMMAZZA e tutti gli altri sinonimi violenti vengano ora gestiti dall'azione `FireAt` invece che dalla azione `attack` che è quella definita dalla libreria standard. Facciamo questo in modo che , nel mercato, UCCIDI GESSLER e AMMAZZA WALTER abbiano lo stesso sfortunato risultato di SPARA A GESSLER o MIRA A WALTER.

Talk

L'ultima azione che definiamo - `Talk` - fornisce un semplice sistema di conversazioni già pronte, un sostituto più semplice delle classiche azioni `Answer`, `Ask` e `Tell`.

Il gestore generico `TalkSub` è basato strettamente su `TellSub` (definita nel file di libreria `verblibm.h`, se siete curiosi), e fa queste cose:

1. Gestisce le frasi PARLA CON ME e PARLA CON ME STESSO.
2. Controlla (a) se la creatura con cui si sta parlando ha una proprietà `life`, (b) se tale proprietà è predisposta a gestire l'azione `Talk` e (c) se l'azione `Talk` restituisce `true`. Se tutti e tre questi controlli riescono, allora `TalkSub` non deve fare altro; se almeno uno di questi fallisce, allora `TalkSub`...
3. Visualizza un generico messaggio di rifiuto "niente da dire".

```
[TYPE]
    [ TalkSub;
        if (noun == player)
            print_ret "Niente di quello che dici può sorprenderti.";
        if (RunLife(noun,##Talk) ~= false) return;
        print_ret "Al momento, non ti viene niente da dire.";
    ];
```

NOTA: quella seconda condizione `(RunLife(noun,##Talk) ~= false)` è complessa e oscura dal momento che utilizza `RunLife` – una routine non documentata interna alla libreria standard – per dare l'azione `Talk` alla proprietà `life` dell'NPC. Abbiamo deciso d'usarla alla stessa maniera in cui la usa l'azione `Tell`, senza preoccuparci troppo di come funziona (anche se sembra che `RunLife` restituisca un qualche valore `true` se la proprietà `life` ha intercettato l'azione, `false` se non lo ha fatto). L'operatore `~=` significa "diverso da...".

La grammatica è normale; notate l'uso di `'c//'` per definire C[11] come sinonimo per parla[12]:

[11] In realtà, facendo questo introduciamo un problema secondario: se il giocatore digita solo C allora la libreria visualizzerà il messaggio "Con chi vuoi parlare?". Per risolvere questo problema

```
[TYPE]
    Verb 'chiacchera' 'conversa' 'c//'
        *   'con' creature                      -> Talk;

    Extend 'parla' first
        *   'a'/'ad'/'all^'/'allo'/'alla'/'al'/
            'agli'/'ai'/'alle'/'con' creature  -> Talk;
```

Ecco il più semplice gestore di `Talk` che abbiamo visto - è quello di Gessler il governatore. Ogni tentativo come PARLA A GESSLER otterrà la risposta "Non hai intenzione di rivolgergli la parola".

```
    life [;
        Talk: print_ret "Non hai intenzione di rivolgergli la parola.";
    ],
```

Il gestore di `Talk` di Walter è solo leggermente più complesso:

```
    life [;
        Talk:
            if (location == mercato)
                print_ret "~Stai calmo, figliolo, e confida in Dio.~";
            else
                print_ret "Spieghi a tuo figlio la tua visione della
                vita.";
    ],
```

E quello di Helga è il più sofisticato (anche se lei non dice poi molto):

```
    frasi_dette 0,       ! per contare gli argomenti di conversazione
    life [;
        Talk:
            self.frasi_dette = self.frasi_dette + 1;
            switch (self.frasi_dette) {
                1:  score = score + 1;
                    print_ret "Ringrazi calorosamente Helga per la
                    mela.";
                2:  score = score + 1;
                    print_ret "~Ci vediamo presto.~";
                default: return false;
            }
    ],
```

Questo gestore usa la proprietà di Helga `frasi_dette` - non una proprietà di libreria, ma una che abbiamo inventato, come le proprietà `centro_piazza.avvertimenti` e `palo.salutato` - per tener conto di quello che è stato detto, permettendo due frammenti di conversazione (e assegnando qualche punto) prima di ricadere nell'imbarazzante silenzio di "Al momento, non ti viene niente da dire".

Questa è la fine della nostra piccola favola; troverete una trascrizione e l'intero sorgente nell'appendice C. E ora è il momento di incontrare Capitan Fato!

---

dovremmo parlare di una routine interna chiamata LanguageVerb – non molto complessa, ma un po' troppo pesante per il nostro secondo gioco.

[12] Notate anche l'uso di `extend` e `first` per ridefinire il verbo `'parla'`, già definito in `ItalianG.h`; in inglese `Tell` e `Talk` sono due verbi distinti, in italiano no. NdT.

10. Capitan Fato: prima!

Per quanto semplici siano, i nostri due giochi hanno coperto la maggior parte delle funzionalità di base di Inform, gettando delle fondamenta abbastanza solide da permettervi di cominciare a creare delle storie semplici. Anche se qualcosa di quello che avete incontrato può sembrarvi ancora senza senso, dovreste essere capaci d'esaminare il codice sorgente di un gioco e farvi un'idea generale di cosa accade.

Adesso creeremo un terzo gioco, per mostrarvi alcune caratteristiche addizionali e darvi ancora del codice d'esempio da analizzare. In "Heidi" abbiamo provato ad andare avanti un passo per volta, spiegando ogni pezzettino di codice che abbiamo inserito nel gioco mentre creavamo gli oggetti in sequenza. In "Guglielmo Tell" avrete notato che abbiamo fatto qualche necessaria divagazione esplicativa a mano a mano che i concetti divenivano più interessanti e complessi. Qui organizzeremo le informazioni in sezioni didattiche logiche, definendo alcuni degli oggetti inizialmente in modo minimo e poi aggiungendo complessità quando servirà. Ancora una volta, questo significa che non potrete compilare il gioco dopo l'aggiunta di ogni spezzone di codice, quindi, se state digitando il codice del gioco a mano a mano che leggete, avrete bisogno di controllare i consigli in "Compilando strada facendo" nell'Appendice D.

Abbiamo già visto una grossa parte di ciò che va a finire nel gioco; potete dedurre da questo che l'attività di progettazione di un gioco è abbastanza ripetitiva e che la maggior parte dei giochi sono, quando ci riferiamo al livello della programmazione, solo un altro remake del solito vecchio tema. Be', sì e no: avete una cinepresa e avete visto come si girano alcuni brevi film, ma da qui a Casablanca ce n'è di strada da fare. Per rimanere su questa analogia, adesso costruiremo la sequenza iniziale di un film indipendente di serie B, un tributo allo stile da super-eroi reso famoso da una intera infanzia di fumetti:

> "Impersonando il tranquillo John Covarth, assistente garzone in una insignificante drogheria, ti FERMI di colpo quando il tuo udito finissimo decifra una chiamata radio della POLIZIA. Un FOLLE sta attaccando la popolazione al Parco Granaio! Devi indossare velocemente il tuo costume da Capitan FATO...!"

cosa che non sarà così semplice da fare. In questo breve esempio, il giocatore vincerà quando riuscirà a cambiarsi nel suo costume da super-eroe e volare via per incontrare il nemico. Il confronto avverrà – forse – in un altro gioco, nel quale potremo solo sperare che Capitan Fato riuscirà ad annientare le forze del male, grazie ai suoi misteriosi (e qui non specificati) poteri.

Dissolvenza: una non meglio descritta strada cittadina

Il gioco comincia con il mite John Covarth che cammina per la strada. Prepariamo il gioco come al solito:

```
[TYPE]
    !% -SD
    !=====================================================================
    Constant Story "Dressed To Save";
    Constant Headline
        "^Semplice esempio in Inform
        ^di Roger Firth and Sonja Kesserich.^";
        ! Traduzione di Paolo Lucchesi
    Release 3;  Serial "040804";    ! conto delle release pubbliche

    Constant MANUAL_PRONOUNS;
    Constant MAX_SCORE     2;
    Constant OBJECT_SCORE  1;
    Constant ROOM_SCORE    1;

    Include "Parser";
    Include "VerbLib";
    Include "Replace";

    !=====================================================================
    ! Classi

    Class Room
        with   description "IN COSTRUZIONE",
        has light;

    Class Appliance
        with   before [;
                    Take,Pull,Push,PushDir:
                        "Anche se i tuoi muscoli SCOLPITI e adamantini ne
                         sarebbero in grado, tu sei contrario ai danni
                         alla altrui proprietà.";
               ],
        has scenery;

    !=====================================================================
    ! Oggetti

    Room strada "In strada"
        with
            name   'citta'      'edifici'      'grattacieli'      'negozi'
                   'appartamenti' 'macchine',
            description
                   "Da una parte, che grazie al tuo SOVRUMANO senso
                    della direzione sai essere il NORD, c'è un bar in
                    cui si può anche pranzare a quest'ora. Verso sud,
                    vedi una cabina del telefono.";

    !=====================================================================
    ! Cosa possiede il giocatore

    !=====================================================================
    ! Entry point routines

        [ Initialise;
            location = strada;
```

```
        lookmode = 2;
        "^^Impersonando il tranquillo John Covarth, assistente
         garzone in una insignificante drogheria, ti FERMI di colpo
         quando il tuo udito finissimo decifra una chiamata radio
         della POLIZIA. Un FOLLE sta attaccando la popolazione al
         Parco Granaio! Devi indossare velocemente il tuo costume da
         Capitan FATO...!^^";
]

!==============================================================
! Standard and extended grammar

Include "ItalianG";
```

Quasi tutto è familiare, a parte qualche dettaglio:

```
Constant MANUAL_PRONOUNS;
Constant MAX_SCORE     2;
Constant OBJECT_SCORE  1;
Constant ROOM_SCORE    1;
```

Inform utilizza in modo predefinito un sistema di pronomi automatici: non appena il personaggio giocante entra in una stanza, la libreria assegna pronomi come ESSO o LUI agli oggetti appropriati (se giocate "Heidi" o "Guglielmo Tell" e digitate PRONOUNS, potete vedere come cambiano le impostazioni). C'è un'altra opzione. Se dichiariamo la costante `MANUAL_PRONOUNS`, forziamo la libreria ad assegnare i pronomi agli oggetti solo quando il giocatore li menziona (cioè, ESSO non viene assegnato fino a che il giocatore digita, per esempio, ESAMINA ALBERO, e a tal punto ESSO diventa l'ALBERO). Il comportamento dell'assegnazione dei pronomi è una questione di gusto personale; nessun sistema è obiettivamente perfetto.

A parte la costante `MAX_SCORE` che abbiamo già visto in "Guglielmo Tell", che definisce il numero massimo di punti che si possono fare, adesso vediamo due nuove costanti: `OBJECT_SCORE` e `ROOM_SCORE`. Ci sono vari sistemi di punteggio predefiniti in Inform. In "Guglielmo Tell" abbiamo visto come potevate manualmente aggiungere (o sottrarre) punti cambiando il valore della variabile **`score`**. Un altro approccio è quello di dare punti al giocatore alla prima occasione in cui (a) entra in una particolare stanza, o (b) raccoglie un particolare oggetto. Per impostare una stanza o un oggetto veramente come "particolari", tutto quello che dovete fare è di dar loro l'attributo **`scored`**; la libreria pensa a tutto il resto. I punteggi predefiniti sono di cinque punti per le stanze finalmente raggiunte e quattro punti per la straordinaria acquisizione di oggetti. Con le costanti `OBJECT_SCORE` e `ROOM_SCORE` possiamo cambiare questi valori base; Per amor d'esempio, abbiamo deciso di dare modestamente un punto per ognuno. A proposito, l'uso del segno di uguaglianza `=` è opzionale con `Constant`; queste due linee hanno identico effetto:

```
Constant ROOM_SCORE   1;
Constant ROOM_SCORE = 1;
```

Un'altra differenza ha a che fare con una speciale scorciatoia che Inform fornisce per far visualizzare stringhe di testo. Finora, vi abbiamo mostrato:

```
    print "E ora, per qualcosa di completamente diverso.^";return true;
    ...
    print_ret "E ora, per qualcosa di completamente diverso.";
```

Entrambe le righe fanno la stessa cosa: visualizzano la stringa tra virgolette, aggiungono un carattere di andata a capo, e restituiscono il valore vero. Come avete visto nei precedenti giochi di esempio, questo succede abbastanza spesso, così esiste un modo ancora più breve per ottenere lo stesso risultato:

```
    "E ora, per qualcosa di completamente diverso.";
```

Cioè, *in una routine* (dove il compilatore si aspetta di trovare una collezione di istruzioni ognuna terminata da un punto e virgola), una stringa tra doppie virgolette da sola, senza bisogno di alcuna parola chiave esplicita, funziona esattamente come se avesse un `print_ret` davanti. Ricordate che questo modo di mostrare il testo implica un `return true` alla fine (che perciò provoca l'uscita immediata dalla routine). Questo dettaglio diviene importante se *non* vogliamo restituire vero dopo che la stringa è stata mostrata sullo schermo - in questo caso dobbiamo usare l'istruzione `print` esplicita.

Noterete che - cosa inusuale per una stanza - il nostro oggetto strada ha una proprietà `name`:

```
    Room    strada "In strada"
        with   name   'citta' 'edifici' 'grattacieli' 'negozi'
                      'appartamenti' 'macchine',
        ...
```

Poiché normalmente non ci si riferisce alle stanze per nome, questo può sembrare strano. In realtà, stiamo illustrando una caratteristica di Inform: la capacità di definire parole di dizionario come "conosciute ma non rilevanti" in questa locazione. Se il giocatore qui digita ESAMINA CITTA', l'interprete risponderà "Non è qualcosa di cui tu abbia bisogno per RISOLVERE la situazione", piuttosto che l'ingannevole "Non vedi nulla del genere". Solitamente preferiamo gestire parole scenografiche del genere utilizzando classi come `Prop` e `Furniture`, ma qualche volta la proprietà `name` di una stanza è una soluzione rapida ed efficace.

In questo gioco, forniamo una classe chiamata `Appliance` per prenderci cura del mobilio e degli oggetti non spostabili. Noterete che la stanza di partenza che abbiamo definito non ha per ora collegamenti. La descrizione menziona una cabina telefonica ed un bar, perciò potremmo volerli programmare. Mentre il bar è una stanza normale, può sembrare logico che la cabina telefonica sia in realtà una grossa scatola sul marciapiede; perciò definiamo un `container` sistemato nella strada, e in cui il giocatore può entrare.

```
[TYPE]
    Appliance cabina "cabina del telefono" strada
        with
            name   'vecchia' 'gialla' 'pittoresca' 'cabina' 'telefono',
            description
                "Il vecchio pittoresco modello giallo, con spazio per una
                 sola persona.",
            before [;
```

```
            Open:  "La cabina è già aperta.";
            Close: "Non c'è modo di chiudere la cabina.";
        ],
        after [;
            Enter:
                "Con velocità implausibile, ti fiondi all'interno
                 della cabina.";
        ],
    has enterable container open female;
```

La parte interessante sono gli attributi alla fine della definizione. Ricorderete dall'oggetto nido di Heidi che un `container` è un oggetto in cui possono essere messi altri oggetti. Se rendiamo qualcosa `enterable`, il giocatore conta come uno di questi oggetti, così ci si può schiacciare dentro. Infine, i `container` sono, per definizione, intesi come chiusi. Potete renderli `openable` (apribili) se volete che il giocatore li possa APRIRE e CHIUDERE a piacimento, ma questo comportamento non sembra appropriato per una cabina pubblica, diventerebbe noioso dover digitare APRI CABINA e CHIUDI CABINA quando queste azioni non fanno nulla di speciale, così aggiungiamo invece l'attributo `open` (come facemmo con il nido), dicendo all'interprete fin dall'inizio che il contenitore è aperto. Il giocatore, ovviamente, non è consapevole del nostro arrangiamento; può certamente provare ad APRIRE o CHIUDERE la cabina, perciò intercettiamo queste azioni in una proprietà `before` che semplicemente dice al giocatore che queste non sono scelte rilevanti. La proprietà `after` fornisce un messaggio personalizzato per ignorare il messaggio di default della libreria per comandi come ENTRA NELLA CABINA o VAI NELLA CABINA. Solo un accenno a `female`, l'attributo che specifica il genere dell'oggetto in modo che la libreria scelga gli opportuni articoli da abbinargli (si ricorda che nei giochi in italiano che sfruttano la libreria INFIT tutti gli oggetti sono di default considerati maschili (`male`) se non diversamente specificato come in questo caso).

Poiché nella descrizione della strada abbiamo detto al giocatore che la cabina telefonica è a sud, egli potrebbe anche tentare di digitare SUD. Dobbiamo intercettare questo tentativo e ridirigerlo appropriatamente (già che ci siamo, aggiungiamo un collegamento verso l'ancora indefinita stanza del bar e un messaggio di default per i movimenti che non sono permessi):

```
Room    strada "In strada"
    with
        name   'citta'     'edifici'     'grattacieli'     'negozi'
               'appartamenti' 'macchine',
        description
               "Da una parte, che grazie al tuo SOVRUMANO senso
                della direzione sai essere il NORD, c'è un bar in
                cui si può anche pranzare a quest'ora. Verso sud,
                vedi una cabina del telefono.",
        n_to bar,
        s_to [; <<Enter cabina>>; ],
        cant_go
               "Non c'è tempo per esplorare. Ti muoverai molto più
                velocemente nel tuo costume da Capitan FATO.";
```

Questo per quanto riguarda l'entrare nella cabina. Ma per uscirne? Il giocatore può digitare ESCI o FUORI mentre è dentro a un contenitore `enterable` e l'interprete obbedirà ma, di nuovo, potrebbe digitare NORD. Questo è un problema, perché noi siamo in realtà nella strada (pur se dentro la cabina) e a nord abbiamo il `bar`. Possiamo provvedere a questa condizione nella proprietà `before` della stanza:

```
before [;
    Go:
        if (player in cabina && noun == n_obj) <<Exit cabina>>;
],
```

Visto che siamo all'aperto e che la cabina fornisce un riparo, non è impossibile che il giocatore possa provare semplicemente DENTRO, che è un ragionamento perfettamente valido. Tuttavia, questo sarebbe un comando ambiguo, dal momento che si può anche riferire al bar, quindi esprimiamo la nostra perplessità e forziamo il giocatore a provare qualcos'altro:

```
n_to bar,
s_to [; <<Enter cabina>>; ],
in_to "Va bene, ma da che parte?",
```

Ora sembra essere tutto a posto, eccetto che per un piccolissimo particolare. Abbiamo detto che, mentre è all'interno della cabina, la posizione del personaggio giocante è sempre nella locazione `strada`, senza riguardo del fatto che si trova all'interno di un oggetto `container`; se il giocatore provasse a scrivere GUARDA, riceverebbe il messaggio:

```
In strada (nella cabina del telefono)
"Da una parte, che grazie al tuo SOVRUMANO senso della direzione
sai essere il NORD, c'è un bar in cui si può anche pranzare a
quest'ora. Verso sud, vedi una cabina del telefono.";
```

Descrizione difficilmente appropriata quando si è *dentro* la cabina. Esistono più modi di risolvere il problema, a seconda del risultato che si vuole ottenere. La libreria fornisce una proprietà chiamata `inside_description` che potete utilizzare con i contenitori in cui si può entrare. Funziona più o meno come la proprietà `description` normale, ma viene stampata a schermo solo quando il giocatore si trova all'interno del `container`. La libreria fa uso di questa proprietà in una maniera molto ingegnosa, poiché per ogni azione GUARDA controlla se possiamo vedere al di fuori del contenitore: se il contenitore ha l'attributo `transparent` impostato, o se è `open`, la libreria mostra prima la normale `description` della stanza e poi la `inside_description` del contenitore. Se la libreria decide che non possiamo vedere fuori del contenitore, viene mostrata solo la `inside_description`. Guardate ad esempio il seguente esempio (semplificato):

```
Room  palcoscenico "Sul palcoscenico"
   with   description
       "Il palcoscenico è pieno degli aggeggi magici di David
        Copperfield.",
...
```

```
Object scatola_magica "scatola magica" palcoscenico
    with   description
               "Una grossa scatola allungata decorata con stelle
                argentate, dove fanciulle succintamente abbigliate
                eseguono un numero della scomparsa della donna.",
           inside_description
               "I pannelli interni della scatola magica sono coperti
                di velluto nero. C'è un minuscolo interruttore
                accanto al tuo piede destro."
           ...
    has container openable enterable light female;
```

Adesso, il giocatore potrebbe APRIRE LA SCATOLA ed ENTRARE NELLA SCATOLA. Un giocatore che provasse a GUARDARE riceverebbe il messaggio:

```
Sul palcoscenico (nella scatola magica)
Il palcoscenico è pieno degli aggeggi magici di David Copperfield.

I pannelli interni della scatola magica sono coperti di velluto
nero. C'è un minuscolo interruttore accanto al tuo piede destro.
```

Se ora il giocatore chiude la scatola e GUARDA:

```
Sul palcoscenico (nella scatola magica)
I pannelli interni della scatola magica sono coperti di velluto
nero. C'è un minuscolo interruttore accanto al tuo piede destro.
```

Nel nostro caso, però, noi vogliamo che la descrizione della strada non sia mostrata per niente (anche se una persona che telefona dovrebbe in teoria poter vedere la strada dall'interno di una cabina telefonica). Il problema è che abbiamo reso la cabina un contenitore `open`, e in questo modo la descrizione della strada verrebbe mostrata sempre. C'è un'altra soluzione. Possiamo rendere la proprietà `description` della stanza `strada` un po' più complessa, e cambiarne il valore: invece di una stringa, scriviamo una routine incorporata. Ecco la stanza (quasi) finita:

```
[TYPE]
    Room    strada "In strada"
        with
            name   'citta'     'edifici'     'grattacieli'     'negozi'
                   'appartamenti' 'macchine',
            description [;
                if (player in cabina)
                    "Da questo punto STRATEGICO ottieni una visuale
                     completa di tutto il marciapiede e dell'ingresso al
                     bar di Benny.";
                else
                    "Da una parte, che grazie al tuo SOVRUMANO senso
                     della direzione sai essere il NORD, c'è un bar in
                     cui si può anche pranzare a quest'ora. Verso sud,
                     vedi una cabina del telefono.";
            ],
            before [;
                Go:
                    if (player in cabina && noun == n_obj)
                        <<Exit cabina>>;
            ],
            n_to [; <<Enter fuori_dal_bar>>; ],
            s_to [; <<Enter cabina>>; ],
```

```
        in_to "Va bene, ma da che parte?",
        cant_go
            "Non c'è tempo per esplorare. Ti muoverai molto più
             velocemente nel tuo costume da Capitan FATO.";
```

La descrizione dall'interno della cabina nomina il marciapiede, cosa che potrebbe invitare il giocatore a ESAMINARLO. Nessun problema:

```
[TYPE]
    Appliance "marciapiede" strada
        with   name 'marciapiede' 'selciato' 'strada',
               article "il",
               description
                   "Esegui un veloce controllo del marciapiede e scopri,
                    con tua immensa sorpresa, che è in TUTTO simile ad
                    ogni altro marciapiede della CITTA'!";
```

Purtroppo, entrambe le descrizioni menzionano anche il bar, che sarà una locazione e perciò un oggetto non esaminabile dall'esterno. Il giocatore può entrarci e riceverà qualunque descrizione scriveremo, come risultato di un'azione GUARDA (che avrà a che fare con come il bar appare *dall'interno*); ma mentre siamo in strada abbiamo bisogno di qualcos'altro per descriverlo:

```
[TYPE]
    Appliance fuori_dal_bar "Il bar di Benny" strada
        with
            name 'bar' 'di' 'benny' 'locale' 'entrata',
            description
                "Il miglior bar della città per uno spuntino veloce. Il
                 bar di Benny ha un look da nave spaziale anni 50'",
            before [;
                Enter:
                    print  "Con un impressionante commistione di fretta e
                            nonchalance entri all'interno del bar.^";
                    PlayerTo(bar);
                    return true;
            ],
        has enterable proper;
```

NOTA: Se volessimo chiamare il nostro bar "caffè", sarebbe consigliabile all'interno del gioco semplificarlo in "caffe". Questo per chiarezza, e non perché Inform non supporti le lettere accentate. L'*Inform Designer's Manual* spiega in dettaglio come mostrare questi caratteri nel paragrafo "§1.11 *How text is printed*" e fornisce l'intero elenco dei caratteri della Z-Machine nella Tabella 2. Nel nostro caso, avremmo potuto mostrare questo:

```
Il miglior caffè della città per uno spuntino veloce. Il caffè di
Benny ha un look da nave spaziale anni 50'.
```

impostando la proprietà `description` come una qualsiasi di queste:

```
description
    "Il miglior caff@`e della città per uno spuntino veloce. Il
    caff@`e di Benny ha un look da nave spaziale anni 50'",
```

```
description
    "Il miglior caff@@182 della città per uno spuntino veloce. Il
    caff@@182 di Benny ha un look da nave spaziale anni 50'",
```

```
description
    "Il miglior caff@{E8} della città per uno spuntino veloce. Il
    caff@{E8} di Benny ha un look da nave spaziale anni 50'",
```

Tuttavia, tutte e tre le forme sono più difficili da leggere del semplice "caffe", perciò è meglio rendersi la vita semplice.

NOTA: la specifica della nota precedente seppur non comune per gli anglofoni, rappresenta la normalità per l'italiano che fa un uso frequente delle lettere accentate. Al momento in cui scriviamo il compilatore di Inform non sembra presentare problemi nella compilazione di testi in cui si fa uso dei caratteri accentati normali (àèéìùò). Allo stesso modo gli interpreti moderni non hanno difficoltà nella gestione dei caratteri accentati. Per tale motivo nei codici di questo volume non abbiamo usato tali forme di codifica delle lettere accentate. Nel caso doveste ricevere qualche errore, magari perché avete deciso di usare un compilatore o un interprete più datato, potete usare uno dei su menzionati metodi (riportiamo la tabella dei codici anche in appendice a questo volume).

A differenza dell'oggetto marciapiede, stavolta offriamo più di una semplice descrizione. Poiché il giocatore potrebbe tentare di ENTRARE NEL BAR come ragionevole via d'accesso - cosa che avrebbe confuso immensamente l'interprete - cogliamo l'opportunità per rendere anche quest'oggetto `enterable`, e bariamo un po'. L'attributo `enterable` ha permesso al verbo ENTRA di essere applicato a quest'oggetto, ma questo non è un `container`; vogliamo invece che il giocatore sia spedito nella vera stanza del bar. Questa cosa è gestita dalla proprietà `before`, che intercetta l'azione, mostra un messaggio e teletrasporta il giocatore dentro al bar. Facciamo `return true` per informare l'interprete che ci siamo presi cura da soli dell'azione `Enter`, e che può fermarsi lì.

Come ultimo dettaglio, notate che adesso abbiamo due modi di entrare nel bar: la proprietà `n_to` della stanza strada e l'azione `Enter` dell'oggetto `fuori_dal_bar`. Un perfezionista potrebbe far notare che sarebbe più ordinato gestire il reale movimento del giocatore tutto in un unico posto del nostro codice, perché questo aumenta la chiarezza. Per ottenere ciò, ridirigiamo la proprietà `n_to` della strada in questo modo:

```
[TYPE]
    n_to [; <<Enter fuori_dal_bar>>; ],
```

Potreste pensare che questa sia solo una mania inutile, ma a buon intenditor poche parole: in un gioco grande, vorrete che la gestione di un'azione avvenga quanto più possibile in un solo posto, perché vi aiuterà a tener conto di dove le cose stanno accadendo se qualcosa non va *come dovrebbe* (come, credeteci, succederà; leggete "Fare il debug del vostro gioco" Capitolo 16). Non c'è bisogno di essere pignoli, soltanto attenti.

Una cabina telefonica in una situazione del genere è un chiaro invito per il giocatore ad entrarci dentro ed a provare a cambiarsi nel costume di Capitan Fato. Non permetteremo che ciò accada - il giocatore non è Clark Kent, dopo

tutto; più avanti spiegheremo come vietare quest'azione - e questo costringerà il giocatore ad entrare nel bar, cercando un posto tranquillo per spogliarsi; ma prima, congeliamo John Covarth davanti al bar di Benny e riflettiamo su una verità fondamentale.

Un eroe non è una persona normale

Che sarebbe come dire, le azioni normali non saranno le stesse per lui.

Come avrete probabilmente dedotto dai capitoli precedenti, alcune delle azioni predefinite della libreria sono meno importanti di altre nel far avanzare il gioco verso una delle sue conclusioni. La libreria definisce PREGA e CANTA, per esempio, che sono di poca importanza in una normale situazione di gioco; ognuna mostra un messaggio generico, non compromettente a sufficienza, e questo è tutto. Chiaramente, se il vostro gioco include un portale magico che appare solo se il giocatore intona un brano di Wagner, potreste intercettare l'azione `Sing` (canta) in una proprietà `before` e modificare il suo, abbastanza inutile, comportamento predefinito. Altrimenti, troverete il messaggio "Sei stonato come una campana." pronto per voi.

Tutte le azioni, utili o no, hanno una serie di messaggi ad esse associati (i messaggi sono immagazzinati nel file `italian.h` e sono elencati (per la versione inglese) nell'Appendice 4 dell'*Inform Designer's Manual*). Abbiamo già visto un modo per alterare la descrizione del personaggio giocante – "Hai sempre lo stesso bell'aspetto di sempre" - in "Guglielmo Tell", ma anche gli altri di default possono essere ridefiniti a seconda dei vostri gusti e necessità di circostanza.

John Covarth, alias Capitan Fato, potrebbe tranquillamente andar bene per la maggior parte di questi messaggi di default, ma riteniamo che valga la pena di dargli qualche risposta personalizzata. Se non altro, questo migliora l'atmosfera generale, una finezza che molti giocatori considerano importante. Per questo obiettivo, faremo uso dell'oggetto `LibraryMessages`.

```
[TYPE]
    Include "Parser";
    ! devono essere definiti tra  Parser e VerbLib:
    Object  LibraryMessages
        with
            before [;
                Buy:   "Il piccolo commercio ti ha raramente
                        interessato.";
                Dig:   "I tuoi super-sensi non percepiscono NIENTE sotto
                        terra che possa interessarti in questo
                        momento.";
                Pray:  "Non hai bisogno di disturbare DIVINITA'
                        onnipotenti per risolvere la situazione.";
                Sing:  "Ahimè! Questo non è uno dei tuoi superpoteri.";
                Sleep: "Un eroe è SEMPRE all'erta.";
                Sorry: "Capitan FATO non ha tempo per le scuse, ma solo
                        per l'AZIONE.";
```

```
            Strong: "Un vocabolario non adatto ad un EROE come te.";
            Swim:  "Rivolgi la tua ATTENZIONE alla ricerca di un
                    posto adatto per ESERCITARE il tuo superiore
                    stile di nuoto ma, ahimè, non trovi niente di
                    simile.";
            Miscellany:
                if (lm_n == 19)
                    if (vestiti has worn)
                        "Negli abiti della tua identità segreta,
                         riesci in maniera estremamente efficace a
                         sembrare un perdente, un perfetto
                         imbranato.";
                    else
                        "Ora che indossi il tuo costume, proietti
                         l'immagine di PURA potenza , di MUSCOLI gonfi
                         e multicolorati, e di uno stile ARDITO e
                         SOBRIO allo stesso tempo.";
                if (lm_n == 38)
                        "Non hai bisogno di questo verbo per risolvere
                         con SUCCESSO la situazione.";
                if (lm_n == 39)
                        "Non è qualcosa di cui tu abbia bisogno per
                         RISOLVERE la situazione.";
        ];
    Include "VerbLib";
    Include "Replace";
```

Se lo fornite, l'oggetto `LibraryMessages` deve essere definito *tra* l'inclusione di `Parser` e quella di `VerbLib` (altrimenti non funziona e riceverete un errore dal compilatore). Questo oggetto contiene un'unica proprietà - `before` - che intercetta i messaggi di default che volete cambiare. Un tentativo di CANTARE, ad esempio, ora, mostrerà il messaggio "Ahimè! Questo non è uno dei tuoi superpoteri".

In aggiunta a queste risposte specifiche per il singolo verbo, la libreria definisce altri messaggi non direttamente associati a un'azione, come la risposta predefinita quando un verbo non viene riconosciuto, o se tentate di riferirvi a un oggetto che non è nei dintorni, ed altre cose. Si può accedere alla maggior parte di questi messaggi tramite la voce `Miscellany`, che ha una lista numerata di risposte. La variabile `lm_n` contiene il valore corrente del numero del messaggio da stampare, così potete cambiare il messaggio di default con un controllo come questo:

```
if (lm_n == 39)
    "Non è qualcosa di cui tu abbia bisogno per RISOLVERE    la
    situazione.";
```

dove 39 è il numero che sta per il messaggio standard "Non è importante ai fini del gioco" - mostrato quando il giocatore menziona una parola che è elencata nella proprietà `name` di una locazione, come abbiamo fatto per la `strada`.

> Nota: ricordate che quando testiamo valori differenti di una stessa variabile, possiamo usare l'istruzione switch. Per la variabile Miscellany, il seguente codice dovrebbe funzionare abbastanza bene:
>
> ...

```
Miscellany:
    switch (lm_n) {
        19:
            if (vestiti has worn)
                "Negli abiti della tua identità segreta, riesci in
                 maniera estremamente efficace a sembrare un
                 perdente, un perfetto imbranato.";
            else
                "Ora che indossi il tuo costume, proietti l'immagine
                 di PURA potenza , di MUSCOLI gonfi e multicolorati,
                 e di uno stile ARDITO e SOBRIO allo stesso tempo.";
        38:
            "Non hai bisogno di questo verbo per risolvere con
             SUCCESSO la situazione.";
        39:
            "Non è qualcosa di cui tu abbia bisogno per RISOLVERE la
             situazione.";
    }
```

Non sorprende quindi che il messaggio predefinito per quando ci si auto-esamina: "Hai sempre lo stesso bell'aspetto di sempre" compaia all'interno di `Miscellany` - è il numero 19 - così possiamo modificarlo attraverso l'oggetto `LibraryMessages` invece di assegnare, come facevamo in precedenza, una nuova stringa alla proprietà `player.description`. Nel nostro gioco, la descrizione del personaggio giocante ha due stati: con gli abiti normali come John Covarth e con il costume da super-eroe come Capitan Fato; da qui il controllo `if (vestiti has worn)`.

Questa discussione sul cambiare le apparenze del nostro eroe mostra che esistono modi diversi di ottenere lo stesso risultato, situazione che è comune quando si progetta un gioco. I problemi possono essere affrontati da angoli differenti; perché usare una tecnica e non un'altra? Di solito, il contesto fa pendere la bilancia in favore di una soluzione, anche se può capitare che optiate per la soluzione non-così-buona a causa di qualche altro fattore più importante. Non vi scoraggiate; scelte come queste diventano sempre più frequenti (e facili) a mano a mano che la vostra esperienza aumenta.

> NOTA: tornando al nostro esempio, un approccio alternativo potrebbe essere quello di impostare la variabile `player.description` nella routine `Initialise` (come abbiamo fatto con "Guglielmo Tell") sulla stringa "vestiti normali", e cambiarlo più tardi, quando serve. Dopo tutto è una variabile, e potete alterare il suo valore con un'altra istruzione come `player.description = look che vi pare` in qualunque punto del codice. Questa soluzione alternativa potrebbe essere migliore se intendessimo cambiare la descrizione del giocatore molte volte nell'arco del gioco. Poiché pensiamo di avere soltanto due stati, l'approccio `LibraryMessages` andrà benissimo.

Un avvertimento finale: come abbiamo spiegato quando estendevamo la grammatica dei verbi standard, voi *potreste* modificare l'appropriato file della libreria e cambiare tutti i messaggi di default, ma questa non sarebbe una pratica vantaggiosa, perché il vostro file di libreria probabilmente non andrebbe più

bene per il prossimo gioco. Consigliamo caldamente l'uso dell'oggetto `LibraryMessages`.

Se state digitando il codice del gioco a mano a mano che andiamo avanti nella trattazione, probabilmente dovreste leggere la breve sezione su "Compilare strada facendo" nell'Appendice D prima di eseguire una prova di compilazione. Una volta che tutto è a posto, è ora che il nostro eroe entri dentro quell'invitante bar.

11. Capitan Fato: seconda!

Visto dall'interno, il bar di Benny è caldo e accogliente, e pieno di clienti venuti a pranzare. Proveremo a mettere insieme qualche immagine appropriata, ma l'elemento fondamentale della stanza non è il suo arredamento: è la porta che porta al gabinetto - e, forse, alla privacy.

Un'atmosfera casalinga

Il bar di Benny è popolato da clienti che si godono il loro pranzo, così non è un buon posto per cambiare identità. Comunque, il gabinetto a nord sembra promettente, anche se Benny ha delle regole rigide sul suo uso, e la porta sembra essere chiusa.

> NOTA CULTURALE: abbiamo usato il termine "gabinetto" per indicare il bagno, una parola che non si riferisce solo all'artifatto di porcellana bianca, ma anche al locale che lo ospita. Questo doppio uso risulterà importante più avanti.

Definiamo la stanza del bar in questa semplice forma:

```
Room    bar "Il bar di Benny"
    with   description
               "Benny offre la MIGLIORE selezione di pezzi dolci e
                sandwich. I clienti riempiono il bancone dove Benny
                in persona riesce a servire, cucinare e riscuotere
                senza la minima esistazione. Sulla parete nord del
                bar vedi una porta rossa che conduce al gabinetto.";
           s_to strada,
           n_to porta_del_gabinetto;
```

Vedremo meglio l'ultima riga (`n_to porta_del_gabinetto`) più tardi, quando definiremo l'oggetto `porta` che giace tra il bar e il gabinetto.

Abbiamo menzionato un bancone.

```
[TYPE]
    Appliance bancone "bancone" bar
        with   name 'bancone' 'banco',
               article "il",
               description
                   "Il bancone è fatto con una strabiliante LEGA di
                    metalli, a PROVA di briciole e liquidi VERSATI e
                    FACILE da pulire. I clienti si godono i loro
                    spuntini con ESTREMA tranquillità, sicuri sapendo
                    che il bancone può resistere a tutto.",
               before [; Receive: <<Give noun Benny>>; ],
        has supporter;
```

La proprietà `before`, all'apparenza normale, in realtà nasconde una piccola sorpresa. Oramai dovreste sentirvi completamente a vostro agio con l'uso della proprietà `before` per intercettare una azione diretta all'oggetto in questione; per esempio, se il giocatore digita COLPISCI BANCONE allora la proprietà `before`

del bancone è potenzialmente in grado di intercettare l'azione `Attack` generata dal comando. Un comando come METTI LA CHIAVE SUL BANCONE, invece, genera *due* azioni. La prima, un'azione `PutOn` (metti su) è inviata alla `chiave` (dicendogli in effetti, vuoi essere posata sopra il bancone?); e questa è la parte normale che già conosciamo. E quindi la sorpresa: viene inviata un'azione `Receive` (ricevi) al `bancone` (dicendogli in effetti, sei felice se una chiave viene posata su di te?). Entrambe le azioni hanno la stessa opportunità di restituire `false` e lasciar continare l'azione, o di restituire `true` e prevenirla.

L'azione `Receive` è generata dalla libreria nell'intestazione dell'azione `PutOnSub`, e anche in `InsertSub` (pertanto un comando come METTI L'UCCELLO NEL NIDO manda un'azione Receive all'oggetto `nido`). Esiste una corrispondente azione `LetGo` (lascia andare), generata dalla libreria da comandi come PRENDI LA CHIAVE DAL BANCONE e TOGLI L'UCCELLO DAL NIDO.

`Receive` e `LetGo` sono esempi di quelle che vengono chiamate **false azioni** (fake action).

> NOTA: in "Guglielmo Tell" abbiamo definito una faretra, tornate al paragrafo "Gli oggetti del giocatore" al Capitolo 7, come un contenitore aperto. Allo stato delle cose, il giocatore può mettere qualsiasi oggetto che possiede, anche inappropriato, nella `faretra`. Avremmo potuto intercettare l'azione `Receive` nella `faretra` per assicurarci che gli unici oggetti accettati dal contenitore fossero le `frecce` (ricordate che ~~, letto come "not", trasforma `true` in `false` e vice versa):
>
> ```
> before [;
>     Drop,Give:
>         print_ret "E' un regalo di Hedwig, tua moglie.";
>     Receive:
>         if (~~(noun ofclass Arrow))
>             print_ret "Solo frecce nella tua faretra.";
> ],
> ```

Qui intercettiamo qualsiasi tentativo di porre un oggetto sul bancone, e lo traduciamo in un tentativo di dare l'oggetto a Benny. Parte della trama del gioco dipende dalla restituzione della chiave del bagno, e dall'acquisto di una deliziosa tazza di cappuccino di fama mondiale. Mettere la chiave ed i soldi sul bancone è un'alternativa al dare tali oggetti direttamente a Benny, piuttosto ragionevole per il giocatore.

Abbiamo menzionato anche dei clienti. Questi sono trattati come NPC, che reagiscono alle performance del nostro eroe.

```
[TYPE]
    Object clienti "clienti" bar
        with
            name 'clienti' 'persone' 'cliene' 'gente' 'uomini' 'donne',
            description [;
                if (costume has worn)
                    "La maggior parte sembra concentrarsi sul proprio
                     cibo, ma alcuni ti osservano blaterando. Deve essere
                     colpa dei colori IPNOTIZZANTI-INSTUPIDENTI del tuo
                     costume.";
```

```
        else
            "Un gruppo di INERMI e IGNARI mortali, gli stessi che
             Capitan FATO ha giurato di DIFENDERE il giorno in
             cui i suoi genitori si sono soffocati con una
             MALIGNA fetta di TORTA DI MIRTILLI.";
    ],
    life [;
        Ask,Tell,Answer:
            if (costume has worn)
                "La gente sembra NON FIDARSI dell'asspetto del
                 tuo FAVOLOSO costume.";
            else
                "Come John Covarth, sei MENO interessante del
                 cibo di Benny.";
        Kiss:
            "Non saprei dirti quali tipi di batteri MUTANTI
             questi STRANIERI stanno portando.";
        Attack:
            "L'insensato massacro di civili è una caratteristica
             dei CATTIVI. Si SUPPONE che tu protegga le persone
             come queste.";
    ],
    orders [;
        "Queste persone non sembrano essere cooperative.";
    ],
    numero_di_commenti 0,   ! contiamo i commenti dei clienti
    daemon [;
        if (location ~= bar) return;
        if (self.numero_di_commenti == 0) {
            self.numero_di_commenti = 1;
            print  "I clienti guardano il tuo costume con aperta
                    curiosità";
        }
        if (random(2) == 1) { ! esegui questo il 50% delle volte
        self.numero_di_commenti = self.numero_di_commenti + 1;
        switch (self.numero_di_commenti) {
               2:  "^~Non sapevo ci fosse il circo in città,~
                    dice un clienti ad un altro. ~Sembra che i
                    pagliacci abbiano il giorno libero.~";
               3:  "^~Questi stilisti non sanno più che fare per
                    farsi conoscere,~ sbuffa un signore
                    corpulento guardando nella tua direzione.
                    Quelli che hanno sentito cercano di
                    nascondere il sogghigno.";
               4:  "^~Deve essere di nuovo carnevale,~ dice un
                    uomo a sua moglie, che sogghigna dandoti
                    un'occhiata di sfuggita. ~Come vola il
                    tempo...~";
               5:  "^~La cosa peggiore delle grandi città~,
                    commenta qualcuno parlando con il suo
                    compagno di tavolo, ~è che vedi gli insetti
                    più schifosi uscire dai cessi.~";
               6:  "^~VORREI davvero poter andare al lavoro in
                    pigiama,~dice una ragazza in tailleur ai suoi
                    colleghi. ~È COSÌ comodo.~";
              default: StopDaemon(self);
            }
        }
    ],
has scenery animate pluralname;
```

Andiamo passo passo. Il nostro eroe entra nel bar vestito come John Covarth, forse riuscirà a cambiarsi nel gabinetto, e quindi dovrà attraversare nuovamente il bar per raggiungere la strada e risolvere il gioco. La descrizione (`description`) dei clienti prende in considerazione cosa sta indossando il giocatore.

In "Guglielmo Tell" abbiamo visto un piccolo esempio della proprietà `life`, ma ora la estenderemo un po'. Come abbiamo già spiegato, `life` vi permette di intercettare quelle azioni proprie degli oggetti animati. Qui noi intercettiamo `Attack` e `Kiss` per offrire qualche messaggio personalizzato per queste azioni quando vengono rivolte verso i clienti. Ancora una volta evitiamo ogni conversazione intercettando `Ask`, `Tell` e `Answer`, in modo da produrre un messaggio che dipende dall'abbigliamento del personaggio.

Un'altra caratteristica degli oggetti animati è la possibilità di ricevere ordini: BILL, AGITA LA LANCIA o ANNIE, PRENDI LA PISTOLA. Queste azioni sono gestite con la proprietà `orders` e, così come avviene per la proprietà `life`, la routine incorporata può diventare parecchio complessa se vogliamo che i nostri NPC si comportino in modo interessante. In questo caso, non abbiamo bisogno che gli avventori svolgano alcun compito per noi, così invece forniamo un semplice messaggio di diniego, giusto nel caso in cui il giocatore provi a dare ordini in giro.

Questo ci lascia soltanto la parte relativa al **demone** (`daemon`). Un demone è una proprietà usata normalmente per eseguire azioni temporizzate o ripetitive senza il bisogno della diretta interazione da parte del giocatore; ad esempio, macchine che funzionano da sole, animali che si muovono, o persone che vanno dietro ai propri affari. In modo più efficace, un demone può tener conto delle decisioni del giocatore in un cento momento, permettendo un comportamento interattivo; questa è, comunque, una caratteristica avanzata che non useremo in questo esempio. Un demone ha la sua opportunità di fare qualcosa alla fine di ogni turno, e tipicamente opera sull'oggetto o con l'oggetto a cui è associato. Nel nostro esempio, il demone produce commenti maligni e ironici da parte dei clienti una volta che il nostro eroe è uscito dal gabinetto vestito con il costume da Capitan Fato.

Per creare un demone, avete bisogno di tre cose:

1. Anzitutto definite una proprietà `daemon` nel corpo dell'oggetto; il valore di questa proprietà è sempre una routine incorporata.
2. Comunque i demoni non fanno nulla fino a quando non li attivate. Questo compito è facilmente svolto dall'istruzione `StartDaemon(identificatore_oggetto)`, che può essere messa in qualsiasi punto (se volete che il demone di un certo oggetto sia attivo dall'inizio del gioco, mettete l'istruzione nella routine `Initialise`).
3. Quando il demone ha svolto il suo compito (se ciò accade), potete fermarlo con l'istruzione `StopDaemon(identificatore_oggetto)`.

Come funziona questo demone in particolare? L'apparizione del nostro eroe, vestito con il suo completo anti-crimine, farà sì che i clienti si sbizzarriranno in commenti irriverenti. Ciò deve accadere nella locazione `bar` – ovvero il posto dove sono gli avventori – pertanto abbiamo bisogno di accertarci che il demone faccia qualcosa di interessante solo mentre il giocatore è nel posto giusto (e non è tornato ad esempio indietro nel gabinetto):

```
if (location ~= bar) return;
```

Pertanto se la locazione non è la stanza `bar` (ricordate che ~= significa "non uguale a"), viene invocata l' istruzione `return` senza aggiungere niente altro; in questo caso il demone non deve fare nulla. Usiamo una istruzione `return` vuota perché il valore ritornato dal demone non ha importanza.

Abbiamo definito una proprietà locale personalizzata, `numero_di_commenti`, per controllare la sequenza delle frasi ironiche degli avventori. Quando il Capitano entra nel stanza del bar dal gabinetto per la prima volta, il valore della proprietà dovrebbe essere zero, così il blocco di istruzioni di questo test:

```
if (self.numero_di_commenti == 0) {
    self.numero_di_commenti = 1;
    print  "I clienti guardano il tuo costume con aperta curiosità";
}
```

verrà eseguito solo una volta. Il nostro intento è quello di visualizzare il testo "I clienti…" subito dopo l'appariscente entrata in scena del nostro eroe, dando la base per i commenti che stanno per arrivare. Dal momento che assegniamo il valore 1 alla proprietà, il messaggio non verrà visualizzato ancora. Notate che abbiamo usato una istruzione `print` esplicita; l'esecuzione del demone continuerà normalmente alla linea successiva.

Inoltre vogliamo che gli avventori facciano commenti ironici quando vedono il Capitano in costume, ma non in modo completamente prevedibile.

```
if (random(2) == 1) ...
```

`random` è una routine di Inform usata per generare numeri casuali o per scegliere casualmente tra più possibilità; nella forma `random(`*`espressione`*`)` restituisce un numero casuale tra 1 e il valore dell'*espressione* tra parentesi (inclusa). Così la nostra condizione attualmente dice: se un valore casuale scelto tra 1 e 2 vale 1, allora esegui qualcosa. Ricordatevi che un demone viene eseguito una volta alla fine di ogni turno, così la condizione prova a tirar fuori un commento da un avventore approssimativamente un turno sì e uno no.

Procediamo, quindi, come abbiamo già visto in "Guglielmo Tell", con un'istruzione `switch` per stampare in ordine i messaggi della sequenza usando con accortezza una nostra proprietà locale, `numero_di_commenti`. Abbiamo scritto solo cinque messaggi (potevano essere cento o uno solo) e dopo raggiungiamo il caso generico `default`, che è un buon posto per fermare il demone, visto che non abbiamo più commenti salaci dei clienti da visualizzare.

Ah, ma quando *inizia* a funzionare il demone? Appena il nostro protagonista esce fuori dalla toilette vistito nel suo pigiama multicolore da supereroe. Siccome vogliamo minimizzare i possibili stati del gioco, adotteremo qualche regola generale per evitare problemi: (a) il giocatore può cambiarsi soltanto nel gabinetto, (b) non permetteremo al giocatore di reindossare i suoi abiti normali, e (c) una volta che il giocatore riesce a uscire in strada con indosso il costume, il gioco è vinto. Così possiamo ragionevolmente supporre che se il giocatore entra nel bar con il costume addosso, sicuramente proviene dal gabinetto. Come conseguenza di tutto ciò, possiamo cambiare la descrizione del bar aggiungendo una proprietà `after` alla locazione del bar:

```
[TYPE]
    Room    bar "Il bar di Benny"
        with   …
            appenauscito false,  ! Prima apparizione di Capitan Fato?
            after [;
                Go: !il giocatore è appena arrivato, viene dal bagno?
                    if (noun ~= s_obj) return false;
                    if (costume has worn && self.appenauscito == false) {
                        self.appenauscito = true;
                        StartDaemon(clienti);
                    }
            ],
            s_to strada;
            n_to porta_del_gabinetto
```

Ci sono due tecniche utili per determinare quando il giocatore sta entrando o uscendo da un stanza. Vedremo più avanti nel dettaglio come comportarsi con un giocatore che tenta di uscire e come impedirglielo se necessario. Per adesso, diciamo solo che, in entrambi i casi, bisogna intercettare l'azione `GO` nell'oggetto locazione; se intercettate l'azione in una proprietà `before`, state controllando per le partenze dalla stanza; se intercettate l'azione in una proprietà `after`, state controllando per gli arrivi nella stanza. In questa occasione desideriamo sapere se il giocatore arriva dal gabinetto, pertanto usiamo una proprietà `after`.

La prima linea:

```
    if (noun ~= s_obj) return false;
```

dice all'interprete che vogliamo fare qualcosa se il giocatore entra nella stanza digitando il comando VAI A SUD (ciò comumente vuol dire "proveniente da nord", ma ricordate che niente vi impedisce di collegare le locazioni senza alcun criterio cardinale); l'interprete applicherà le regole normali per le altre direzioni disponibili.

Quindi controlliamo se il personaggio giocatore sta indossando il costume, nel qual caso facciamo partire il demone dell'oggetto `clienti`. L'uso della proprietà locale `appenauscito` assicura che la condizione sia vera solo una volta, così il blocco di istruzioni allegate ad essa sarà eseguito una volta sola.

Abbiamo finito con i clienti nel bar. Ora abbiamo il gabinetto a nord che, per ragioni di gioco *e* di decenza, è protetto da una porta.

Una porta da adorare

Gli oggetti porta richiedono certe specifiche proprietà e certi specifici attributi. Anzitutto creiamo una porta semplice:

```
Object  porta_del_gabinetto "porta del gabinetto" bar
    with  name 'porta' 'rossa' 'del' 'gabinetto' 'bagno',
        description
            "Una porta rossa con le inequivocabili silhouette di un
             uomo e di una donna che segnano l'ingresso ai locali
             igienici. C'è una nota scarabocchiata attaccata alla
             superficie della porta.",
        door_dir n_to,
        door_to gabinetto,
        with_key chiave_del_gabinetto,
    has scenery door openable lockable locked female;
```

Troviamo questa porta nel bar. Dobbiamo specificare in che direzione la porta conduce, ovvero, come abbiamo specificato nella descrizione del bar, nord. La proprietà `door_dir` serve proprio a questo, e in questo caso prende il valore della proprietà di direzione nord, `n_to`. Dopo dobbiamo dire a Inform qual è la stanza che si trova dietro la porta, cosa che facciamo attraverso la proprietà `door_to` che assume il valore `gabinetto`: stanza che andrà definita in seguito. Ricordate la connessione del bar verso nord, `n_to porta_del_gabinetto`? Grazie ad essa, Inform saprà che la porta blocca la strada in quella direzione e, grazie alla proprietà `door_to`, anche cosa si trova oltre la porta.

Le porte *devono* avere l'attributo `door`, ma oltre a quello abbiamo anche una quantità di opzioni che ci aiutano a definire esattamente con che tipo di porta abbiamo a che fare. Così come i contenitori, le porte possono essere apribili (attributo `openable`, che attiva i verbi APRI e CHIUDI in modo che possano essere applicati all'oggetto) e, dal momento che di default esse sono chiuse, se volete potete dargli l'attributo `open` (aperto). Inoltre, le porte possono essere bloccabili (cioè chiudibili a chiave, grazie all'attributo `lockable` che attiva i verbi BLOCCA/SBLOCCA e le frasi APRI...CON/CHIUDI...CON) e possono essere rese bloccate (attributo `locked`) visto che di default non lo sono. I verbi BLOCCA, SBLOCCA, APRI...CON e CHIUDI...CON hanno bisogno di un oggetto chiave che funzioni con la porta. Questo viene definito con la proprietà `with_key`, il cui valore deve essere l'identificatore interno della chiave; nel nostro esempio, è l'oggetto `chiave_del_gabinetto`, che definiremo in seguito. Se non fornite questa proprietà il giocatore non sarà in grado di bloccare e sbloccare la porta[13].

[13] E bene qui porre l'attenzione del lettore sulla differenza linguistico/culturale tra italiano ed inglese, tenendo a mente che le librerie di Inform sono nate per gli anglofoni e solo poi sono state adattate all'italiano. In particolare per quanto riguarda le porte per gli Inglesi l'aprire o chiudere una porta (open/close) è una azione diversa dal sbloccare/bloccare una porta (lock/unlock). Pertanto di default per aprire una porta chiusa a chiave un inglese distingue e trova perfettamente logico (seppur noioso) digitare due comandi distinti prima UNLOCK DOOR e OPEN DOOR. Un italiano invece troverebbe quantomeno curioso dover digitare una seconda volta un comando

Questa semplice definizione di porta ha un problema, ovvero, esiste soltanto nel bar. Se volete che la porta sia presente anche nel gabinetto potete: (a) definire un'altra porta nel `gabinetto`, o (b) rendere questa porta una porta a doppia faccia.

La soluzione (a) sembra semplice a prima vista, ma poi avrete il problema di tenere lo stato delle due porte (aperta/chiusa, bloccata/sbloccata) in sincronia. In questo scenario, dove potete accedere al gabinetto solo attraverso questa porta, non dovrebbe essere troppo complicato, visto che potete lasciare la porta che si trova nel bar sempre aperta, indipendentemente da quello che il giocatore fa con la porta che si trova nel gabinetto, e viceversa - le due porte non verranno mai viste nello stesso momento. In termini più generali però l'inconsistenza tra le due porte comporta solo problemi; nella maggior parte dei casi è meglio ignorare la soluzione (a).

La soluzione (b) è migliore, visto che avete solo un oggetto porta da gestire, e il suo stato influenza entrambe le sue facce. Comunque il codice diventa un po' più complicato, e dovete definire delle routine per la maggior parte delle proprietà:

```
[TYPE]
    Object  porta_del_gabinetto "porta del gabinetto"
        with
            name 'porta' 'rossa' 'del' 'gabinetto' 'bagno',
            description [;
                if (location == bar)
                    "Una porta rossa con le inequivocabili silhouette di
                     un uomo e di una donna che segnano l'ingresso ai
                     locali igienici. C'è una nota scarabocchiata
                     attaccata alla superficie della porta.";
                else
                    "Una porta rossa senza alcuna caratteristica
                     NOTEVOLE.";
            ],
            found_in bar gabinetto,
            door_dir [;
                if (location == bar) return n_to;
                else return s_to;
            ],
            door_to [;
                if (location == bar) return gabinetto;
                else return bar;
            ],
            with_key chiave_del_gabinetto,
        has scenery door openable lockable locked female;
```

Anzitutto, la porta ora ha bisogno di una proprietà `found_in`, visto che dovrà trovarsi sia nel `bar` che nel `gabinetto`. La descrizione controlla da quale lato della porta stiamo guardanto - controllando il valore della variabile `location`, che contiene la stanza in cui si trova il giocatore - poiché abbiamo una nota affissa su di un lato, ma non sull'altro. E le proprietà `door_dir` e `door_to`

---

di apertura dopo aver già scritto APRI PORTA CON LA CHIAVE (che corrisponde ad una azione UNLOCK). Torneremo sull'argomento più avanti nella trattazione. NdT.

devono usare lo stesso trucco, poiché vogliamo muoverci verso nord dal bar fino al gabinetto, ma verso sud dal gabinetto al bar.

Adesso il gioco visualizzerà "La porta del gabinetto" ogni volta che ha bisogno di riferirsi a tale oggetto. Sarebbe carino se potessimo far sì che il gioco distinguesse tra "la porta che conduce al gabinetto" e "la porta che conduce al bar", a seconda del lato della porta da cui ci troviamo. Per questo abbiamo bisogno della proprietà `short_name`. Abbiamo già parlato del nome esterno definito nella prima riga di un oggetto:

```
Object porta_del_gabinetto "porta del gabinetto"
```

Quel "porta del gabinetto" sarà il nome visualizzato durante il gioco per riferirsi alla porta. Con lo stesso identico effetto avremmo potuto scrivere:

```
Object porta_del_gabinetto
    with short_name "porta del gabinetto",
```

`short_name` è una proprietà che fornisce il nome esterno di un oggetto, o come stringa o come routine incorporata. Normalmente, gli oggetti mantengono lo stesso nome esterno durante tutto il gioco – e il metodo di tenere l'informazione nella prima riga è perfetto in questo caso - ma se è necessario cambiarlo, è facile scrivere una routine come valore di `short_name`:

```
[TYPE]
    Object  porta_del_gabinetto
        with
            name 'porta' 'rossa' 'del' 'gabinetto' 'bagno',
            short_name [;
                if (location == bar) print "porta verso il gabinetto";
                else print "porta verso il bar";
                return true;
            ],
            description
                    ...
```

Notate il `return true` alla fine della routine. Vi ricorderete che la regola standard dice di "restituire `false` per continuare, `true` per interferire e interrompere la normale esecuzione". Nel caso di `short_name`, "continuare" significa "visualizzare il nome esterno che si trova nella prima riga", cosa che può talvolta tornare utile; ad esempio, potreste scrivere una routine `short_name` per stampare qualcosa da una lista prima del nome esterno - come una splendente/debole/inutile lampada.

> NOTA: cosa viene visualizzato se non c'è un nome esterno nella prima riga dell'oggetto? Se avete letto la sezione "Compilare strada facendo" nell'appendice C, ricorderete che l'interprete usa semplicemente l'identificatore interno tra parentesi; ovvero, senza nome esterno e senza proprietà `short_name`, potremmo vedere:
>
> ```
> Apri la (porta_del_gabinetto).
> ```

E lo stesso principio si applica se dimentichiamo di restituire `false` dalla routine `short_name`: otterremmo prima il risultato della nostra istruzione `print`, e dopo la regola standard visualizzerebbe l'identificatore interno:

```
Apri la porta del gabinetto(porta del gabinetto).
```

Le porte possono diventare più complicate di così (no, per favore, non scagliate questa guida fuori dalla finestra). Ecco un po' di codice opzionale deluxe per rendere la porta un poco più amichevole durante il gioco; potete saltarlo se prevedete un mal di testa.

La nostra porta si comporta bene durante il gioco. Può essere bloccata e sbloccata se il giocatore ha la chiave giusta; può essere aperta e chiusa. Una sequenza di comandi per entrare nel gabinetto e chiudersi la porta alle spalle sarebbe:

APRI LA PORTA CON LA CHIAVE, APRI LA PORTA, VAI A NORD,

CHIUDI LA PORTA, CHIUDI LA PORTA CON LA CHIAVE

Quando abbiamo finito torniamo nel bar:

APRI LA PORTA CON LA CHIAVE, APRI LA PORTA, SUD,

CHIUDI LA PORTA, CHIUDI LA PORTA CON LA CHIAVE

Questo gioco ha una sola porta, ma se ne avesse tre o quattro il giocatore si scoccerebbe (perlomeno) dovendo scrivere così tanti comandi solo per attraversare una porta. Questo è quello che viene normalmente detto "design scarso", perché il gioco si rallenta solo per effettuare una semplice azione che non nasconde segreti o sorprese. Quanto può essere eccitante attraversare una normalissima porta, dopo tutto?

Se qualche linea di codice può rendere la vita più facile per il giocatore, allora ne vale la pena. Aggiungiamo qualche miglioria alla nostra porta del gabinetto nelle proprietà `before` e `after`:

```
[TYPE]
    before [ ks;
        Open:
            if (self hasnt locked || chiave_del_gabinetto notin player)
                return false;
            ks = keep_silent; keep_silent = true;
            <Unlock self chiave_del_gabinetto>; keep_silent = ks;
            return true;
        Lock:
            if (self hasnt open) return false;
            print "(prima chiudi ", (the) self, ")^";
            ks = keep_silent; keep_silent = true;
            <Close self>; keep_silent = ks;
            return false;
    ],
    after [ ks;
        Unlock:
            if (self has locked) return false;
            print "Apri ", (the) self, ".^";
            ks = keep_silent; keep_silent = true;
```

```
        <Open self>; keep_silent = ks;
        return true;
],
```

L'idea di base qui è quella di permettere al giocatore che ha in mano la chiave di eseguire una sola azione per sbloccare e aprire la porta (e quindi anche per chiuderla e bloccarla). Le azioni rilevanti sono `Unlock` e `Open`, e `Lock` (`Close` non è necessario; se il giocatore chiude la porta non possiamo assumere che voglia chiuderla a chiave).

- **Open**: se la porta non è bloccata o il giocatore non è in possesso della chiave, allora procediamo con l'azione `Open` standard definita dalla libreria. Ci rimane il caso della porta bloccata e il giocatore che possiede la chiave, così rimandiamo l'elaborazione all'azione `Unlock`, dando come argomenti la porta (`self`) e la chiave del gabinetto. Siccome usiamo solo una parentesi angolata per lato `<...>`, l'azione riprende dopo che la porta è stata sbloccata (notate che l'azione `Unlock` si prende cura anche di aprire la porta). Finalmente restituiamo `true` per far sì che la libreria non provi ad aprire la porta.
- **Lock**: se la porta è già chiusa, allora procediamo con l'azione `Lock` standard di libreria. Altrimenti diciamo al giocatore che stiamo chiudendo la porta per lui, rimandiamo brevemente l'esecuzione all'azione `Close` per chiuderla e poi restituiamo `false` in modo che l'azione `Lock` proceda normalmente.
- **Unlock**: piazziamo questa azione nella proprietà `after` così (speriamo) l'azione `Unlock` è già accaduta. Se la porta è sempre bloccata allora è successo qualcosa di sbagliato, così restituiamo `false` per visualizzare il messaggio standard per lo sbloccaggio non riuscito. Altrimenti la porta ora è sbloccata, così informiamo il giocatore che stiamo aprendo la porta e rimandiamo l'esecuzione all'azione `Open` per aprirla effettivamente, restituendo `true` per sopprimere il messaggio standard.

In tutto ciò c'è una variabile di libreria chiamata `keep_silent`, che può assumere i valori `false` (lo stato normale) o `true`; quando ha il valore `true`, l'interprete non visualizza il messaggio relativo all'azione in corso, così possiamo evitare cose come:

```
> APRI LA PORTA
Apri la porta del gabinetto.
Sblocchi la porta del gabinetto e la apri.
```

Anche se vogliamo assegnare a `keep_silent` il valore `true` per tutta la durata delle nostre routine, dobbiamo poi risistemarla subito dopo. In un caso come questo è buona pratica preservare il suo valore iniziale (che probabilmente era `false`, ma dovreste sempre evitare presunzioni pericolose); usiamo una variabile locale `ks` per ricordare il valore iniziale, così possiamo ripristinarla dopo. Ricordere che una variabile locale in una routine indipendente è dichiarata tra il nome della routine e il punto e virgola:

```
[ GiaStatoQui questa_stanza;
```

Nello stesso modo, una varabile locale in una routine incorporata è dichiarata tra il simbolo `[` iniziale e il punto e virgola:

```
before [ ks;
```

Potete dichiarare in questo modo – separandole con degli spazi – fino a 15 variabili che saranno usabili solo nella routine incorporata. Quando assegniamo loro un valore, come:

```
ks = keep_silent;
```

stiamo rendendo il valore di `ks` identico a qualsiasi valore abbia `keep_silent` (sia esso `true` o `false`; non ce ne importa). Dopo assegniamo a `keep_silent` il valore `true`, facciamo le nostre azioni silenziose e assegniamo:

```
keep_silent = ks;
```

che riassegna a `keep_silent` il valore originalmente memorizzato in `ks`. L'effetto è che siamo riusciti a lasciare `keep_silent` esattamente come era prima che la cambiassimo.

Bene, questo è tutto a proposito delle porte. Tutto? Beh, no, non esattamente; ogni oggetto può diventare tanto complesso quanto lo permette la vostra immaginazione, ma noi abbandoneremo l'argomento qui. Se volete vedere porte più sofisticate, controllate gli esercizi 3 e 4 nell'*Inform Designer's Manual*, dove una porta si apre e si sblocca da sola se il giocatore si muove nella sua direzione.

Fino ad ora abbiamo il giocatore di fronte ad una porta chiusa a chiave che conduce al gabinetto. Un vicolo cieco? No, la descrizione menziona una nota scarabocchiata attaccata alla porta. Questa non dovrebbe creare problemi:

```
[TYPE]
    Object  "nota" bar
        with
            name 'nota' 'scarabocchiata',
            description [;
                if (self.letta == false) {
                    self.letta = true;
                    "Rivolgi la tua visione a ULTRAFREQUENZA AVANZATA
                     verso la nota e socchiudi gli occhi concentrandoti,
                     arrendendoti solo quando i bordi della nota iniziano
                     ad annerirsi sotto l'incredibile intensità del tuo
                     sguardo infuocato. Rifletti ancora una volta su
                     quanto sarebbe stato utile se tu avessi mai imparato
                     a leggere. ^^Una vecchia signora premurosa si
                     avvicina e ti spiega: ~Devi chiedere la chiave a
                     Benny, al bancone.~^^ Ti volti verso di lei e inizi
                     a dire: ~Oh, la SAPEVO, ma...~^^ ~Di nulla,
                     figliolo,~ dice la signora mentre esce dal bar.";
                }
                else
                    "La nota indecifrabile e annerita non ha più SEGRETI
                     per te ADESSO. Ha!";
            ],
            letta false,   ! il giocatore ha già letto la nota?
            before [;
                Take:
                    "Non hai motivo di raccogliere note INDECIFRABILI.";
            ],
```

```
    has scenery female;
```

Notate soltanto come cambiamo la descrizione dopo la prima volta che il giocatore esamina la nota, usando la proprietà locale `letta` creata proprio per questo scopo. Non vogliamo che il giocatore se ne vada via con la nota, così intercettiamo l'azione `Take` e visualizziamo qualcosa di più adatto del messaggio standard per gli oggetti di scenografia: "È difficilmente trasportabile".

Abbiamo parlato molto della chiave del gabinetto; dovrebbe essere il momento per scrivere il codice che la riguarda. Originariamente la chiave è in possesso di Benny, e il giocatore dovrà chiederla, così come spiega la nota. Anche se definiremo Benny in dettagli nel prossimo capitolo, presentiamo una definizione di base, in modo che la chiave abbia un oggetto genitore.

```
[TYPE]
    Object  benny "Benny" bar
        with
            name 'benny',
            description
                "Un uomo ingannevolmente GRASSO dotato di incredibile
                 agilità, Benny intrattiene i clienti schiacciandosi noci
                 di cocco sulla fronte quando è dell'umore giusto.",
        has scenery animate male proper transparent;


[TYPE]
    Object chiave_del_gabinetto "chiave del gabinetto" benny
        with
            name 'chiave' 'del' 'gabinetto' 'bagno',
            article "la",
            invent [;
                if (vestiti has worn) print "la chiave CRUCIALE";
                else print "la chiave ormai usata e IRRILEVANTE";
                return true;
            ],
            description
                "I tuoi sensi ULTRA-PERCETTIVI non individuano niente di
                 particolare sulla chiave del gabinetto.",
            before [;
                if (self in benny)
                    "SCANDAGLI i l'ambiente con la tua CONSAPEVOLEZZA
                     POTENZIATA, ma non riesci ad individuare alcuna
                     chiave.";
            ],
        has female;
```

Finchè Benny ha la chiave, non c'è logicamente modo di esaminarla (o di compiere qualsiasi azione su di essa), ma non vogliamo che l'interprete sostenga che "Non puoi vedere niente del genere". Abbiamo creato la `chiave_del_gabinetto` come oggetto figlio (child) dell'oggetto `benny`, e potete vedere che Benny ha l'attributo `transparent`; questo significa che la chiave è "in scope" (a portata di mano), e questo autorizza il giocatore a riferirsi ad essa senza che l'interprete se ne lamenti. Siccome Benny ha anche l'attributo `animate`, l'interprete normalmente intercetterebbe l'azione PRENDI LA CHIAVE con un "Sembra appartenere a Benny"; comunque la stessa cosa non

varrebbe per altri comandi come TOCCA LA CHIAVE o ASSAGGIA LA CHIAVE. Così, per prevenire ogni interazione con la chiave fino a che è nelle tasche di Benny, definiamo una proprietà `before`.

```
before [;
    if (self in benny)
        "SCANDAGLI i l'ambiente con la tua CONSAPEVOLEZZA
         POTENZIATA, ma non riesci ad individuare alcuna chiave.";
    Drop:  "Benny si aspetta che tu restituisca la chiave prima o
            poi.";
];
```

Tutte le proprietà `before` che abbiamo creato fino ad ora contenevano una o più etichette per specificare l'azione che dovevano intercettare; ricorderete che in "Guglielmo Tell" abbiamo introdotto l'azione `default` (Vedi "Una classe per le scenografie" nel Capitolo 6) per significare "ogni valore ancora non trattato".

C'è un'altra di quelle etichette qui, per l'azione `Drop`, ma è preceduta da un pezzo di codice che sarà eseguito all'inizio di ogni azione diretta alla chiave. Se sarà ancora in possesso di Benny, visualizzeremo un simpatico rifiuto; se il giocatore possiede la chiave allora evitiamo qualsiasi disposizione e l'azione continua senza senza intoppi.

(Infatti, il cappello sul palo della classe `Prop` introdotto nel Capitolo 8 aveva questa proprietà "escludi ogni azione":

```
before [;
    default:
        print_ret "Sei troppo lontano al momento.";
],
```

Sarebbe stato esattamente lo stesso se avessimo omesso l'etichetta `default`, come abbiamo fatto per la chiave di Benny).

Abbiamo, poi, un'altra piccola innovazione: la proprietà di libreria `invent` (non l'abbiamo fatta noi) che vi permette di controllare come gli oggetti appaiano nell'inventario, piuttosto che nel modo normale. Lasciato a se stesso, l'interprete visualizza semplicemente il nome esterno dell'oggetto, preceduto o da un articolo indeterminativo standard come "un", "una" o "dei", o un articolo specificatamente definito nella proprietà `article`. Qui noi rimpiazziamo "la chiave del gabinetto" con una tra due descrizioni più utili, rendendola l'oggetto più prezioso nelle mani di John Covarth, e qualcosa di cui liberarsi velocemente per Capitan Fato, una volta che non è più di alcuna utilità per lui.

Quando avevamo il giocatore in strada, abbiamo affrontato il problema che egli poteva voler esaminare il bar dall'esterno. Anche se è improbabile che provi ad esaminare il gabinetto dall'esterno, non bisogna sforzarsi più di tanto per fornire un risultato adatto:

```
[TYPE]
    Object  fuori_dal_gabinetto "gabinetto" bar
        with
            name   'gabinetto'  'cesso'  'toilette'  'toilet'  'ritirata'
                   'bagno',
            before [;
```

```
            Enter:
                if (porta_del_gabinetto has open) {
                    PlayerTo(gabinetto);
                    return true;
                }
                else
                    "La tua SUPERBA mente deduttiva comprende che la
                     PORTA è chiusa.";
            Examine:
                if (porta_del_gabinetto has open)
                    "Un pensiero brillante illumina il tuo cervello
                     SUPERLATIVO: una dettagliata esplorazione del
                     gabinetto sarebbe ESTREMAMENTE facilitata se tu
                     entrassi all'interno.";
                else
                    "Con un TREMENDO sforzo di volontà, evochi la tua
                     imperscrutabile VISIONE ASTRALE e la proietti in
                     AVANTI attraverso la porta chiusa... fino a che
                     non ti ricordi che è il Dottor Mystere ad avere
                     i poteri mistici.";
            Open:  <<Open  porta_del_gabinetto>>;
            Close: <<Close porta_del_gabinetto>>;
            Take,Push,Pull:   "Sarebbe PARTE dell'edificio.";
        ],
    has scenery openable enterable;
```

Come per l'oggetto `fuori_dal_bar`, intercettiamo l'azione `Enter`, per teletrasportare il giocatore dentro la stanza gabinetto se egli scrive ENTRA NEL GABINETTO (o per stampare un rifiuto se la porta del gabinetto è chiusa). Il giocatore può provare a scrivere ESAMINA IL GABINETTO; otterrà un differente messaggio se la porta è aperta – lo inviteremo ad entrare - o se è chiusa.

I comandi APRI IL GABINETTO e CHIUDI IL GABINETTO sono rediretti alle azioni `Open` e `Close` della porta del gabinetto; ricordate che le doppie parentesi angolate implicano un'istruzione `return true`, così l'azione si ferma lì e l'interprete non prova a Aprire o Chiudere l'oggetto `fuori_dal_gabinetto` dopo che ha agito sulla porta.

Avete ragione: il gabinetto ha una parte importante in questo gioco (diamo la colpa a infantili influenze materne). Abbiamo introdotto un problema di ambiguità con l'oggetto `fuori_dal_bagno`, e ora abbiamo bisogno d'aiuto per sistemarlo.

12. Capitan Fato: terza!

Usando troppo la parola "gabinetto" ci siamo posti una sfida interessante, e ora vi mostreremo come risolvere le ambiguità che abbiamo introdotto. Inoltre è il momento adatto per sviluppare in pieno l'eponimo proprietario del Bar di Benny.

Troppi gabinetti

Se controllate le proprietà `name` della porta del gabinetto, della chiave del gabinetto e del gabinetto, noterete che la parola del dizionario `'gabinetto'` compare in tutte e tre. Non ci sono problemi se il giocatore menziona le parole PORTA o CHIAVE, ma giungiamo ad uno strano impasse se il giocatore prova a compiere qualche azione usando solo la parola GABINETTO. L'interprete deve pensare velocemente: il giocatore sta parlando della chiave? della porta? del gabinetto? Incapace di decidere, l'interprete chiede "Cosa intendi, la porta che conduce al gabinetto, il gabinetto oppure la chiave del gabinetto?"

Provate a indovinare... Il giocatore non potrà mai riferirsi all'oggetto gabinetto (a meno che non scriva TOILET o RITIRATA, non una scelta ovvia visto che non abbiamo mai usato questi termini in modo che risultino visibili). Se il giocatore risponde GABINETTO il parser avrà sempre tre oggetti con quella parola di dizionario come possibile nome, e quindi chiederà ancora, e ancora - finché non daremo una parola di dizionario che non è ambigua. Un lettore umano sarebbe in grado di capire che la parola GABINETTO da sola si riferisce alla stanza , ma l'interprete no - a meno che non lo aiutiamo un poco.

Potremmo aggirare questo problema in più di un modo, ma approfitteremo di questa opportunità per dimostrare l'uso di una libreria esterna (in gergo, prodotta da terze parti).

Quando sviluppatori esperti trovano un problema che non è facilmente risolvibile, possono venir fuori con una soluzione brillante e poi tenere in conto che altri possono beneficiare dei loro sforzi. Il prodotto di tale generosità prende la forma di un'estensione della libreria accuratamente impacchettata in un file che altri sviluppatori possono incorporare nel loro codice sorgente. Questi file possono essere trovati nell'IF Archive: andate su `http://www.ifarchive.org/indexes/if-archive.html` e poi selezionate `".../infocom"`, `".../compilers"`, `".../inform6"`, `".../library"`, e `".../contributions"`. Tutti questi file contengono del codice Inform. Per usare un'estensione della libreria (detta anche contributo alla libreria , o "library contribution" in Inglese), dovete scaricarla e leggere le istruzioni (normalmente incluse come commento nel file, ma occasionalmente fornite separatamente) per scoprire cosa fare dopo. Normalmente la `Include`-te nel vostro codice (come avete già fatto con `Parser.h`, `VerbLib.h`, `Replace.h` e `ItalianG.h`), ma spesso ci

sono regole sul dove piazzare esattamente l'`Include` nel vostro codice sorgente. Non è insolito trovare altri suggerimenti e avvertimenti.

Per avere aiuto con il nostro problema di ambiguità della parola GABINETTO, andremo ad usare l'estensione di Neil Cerutti `pname.h`, che è stata sviluppata proprio per situazioni come questa. Anzitutto seguiamo il link all'IF Archive e scarichiamo il file compresso `pname.zip`, che contiene due ulteriori file, `pname.h` e `pname.txt`. Mettiamo questi file nella cartella dove stiamo sviluppando il nostro gioco e, se usiamo l'ambiente proposto in "I ferri del mestiere", nella cartella `Inform\Lib\Contrib`. Il file di testo contiene istruzioni per l'installazione e l'uso. In esso troviamo un avviso:

> This version of pname.h is recommended for use only with version 6/10 of the Inform Library
>
> (Si raccomanda di usare questa versione di pname.h solo con la versione 6/10 della libreria Inform)

che non è esattamente la versione che stiamo usando visto che noi abbiamo a disposizione la successiva 6/11, ma questo non sembra comportare problemi[14]. La maggior parte delle estensioni non sono così pedanti, ma `pname.h` traffica con qualche routine nel cuore della libreria standard; queste potrebbero non essere identiche in altre versioni di Inform.

L'introduzione spiega cosa fa esattamente `pname.h`; precisamente, essa vi permette di evitare l'uso di complicate routine `parse_name` per risolvere le ambiguità dell'input del giocatore quando la stessa parola di dizionario si riferisce a più di un oggetto. Una routine `parse_name` sarebbe stata la soluzione del nostro problema se questo file `pname.h` non fosse esistito, ed è decisamente un argomento di programmazione avanzata, difficile da apprendere in un primo approccio. Fortunatamente non dobbiamo preoccuparcene. Neil Cerutti spiega:

Il pacchetto `pname.h` definisce una nuova proprietà per gli oggetti, `pname` (abbreviazione di phrase name), con un aspetto simile a quello della normale proprietà `name`: entrambe contengono una lista di parole del dizionario. Però nella proprietà `pname` l'ordine delle parole è importante, e degli operatori speciali `'.p'` `'.or'` e `'.x'` ti permettono di dare un po' di intelligenza alla lista. Nella maggior parte dei casi in cui la normale proprietà `name` non è abbastanza, potete semplicemente rimpiazzarla con una proprietà `pname`, piuttosto che scrivere una routine `parse_name`.

Vedremo presto come funziona. Guardiamo le istruzioni di installazione:

> Per introdurre questo pacchetto nel vostro programma dovete fare tre cose:

[14] In realtà qualche problema potrebbe sorgere (ancora non riscontrato), nella libreria pname viene replicata la funzione di libreria NounDomain. Funzione che è cambiata tra la libreria 6/10 e 6/11. Su IF Italia dovreste poter scaricare una versione della libreria pname.h perfettamente compatibile con le librerie 6/11 e con il file di testo di spiegazioni allegato tradotto in italiano. Ndt.

1. Aggiungere quattro righe verso l'inizio del programma (prima di includere `Parser.h`)

```
Replace MakeMatch;
Replace Identical;
Replace NounDomain;
Replace TryGivenObject;
```

2. Includere il file `pname.h` subito dopo `Parser.h`

```
Include "Parser";
Include "pname";
```

3. Aggiungere la proprietà `pname` a quegli oggetti che richiedono il riconoscimento delle frasi.

Sembra abbastanza facile. Così, seguendo i passi 1 e 2, aggiungiamo quelle righe `Replace`... prima dell'inclusione di `Parser`, e includiamo `pname.h` subito dopo. `Replace` dice al compilatore che stiamo fornendo dei rimpiazzamenti per alcune routine standard.

```
[TYPE]
    Constant Story "Capitan FATO";
    Constant Headline
             "^Semplice esempio in Inform
              ^di Roger Firth and Sonja Kesserich.^";
              ! Traduzione di Paolo Lucchesi
    Release ;  Serial "040804";    ! conto delle release pubbliche

    Constant MANUAL_PRONOUNS;

    Replace MakeMatch;              ! richiesto da pname.h
    Replace Identical;
    Replace NounDomain;
    Replace TryGivenObject;

    Include "Parser";
    Include "pname";                ! pname.h la trovate nell'Archivio
    ...
```

Adesso il nostro codice sorgente è pronto per beneficiare del pacchetto di libreria. Come funziona? Abbiamo guadagnato una nuova proprietà – `pname` – che può essere aggiunta ad alcuni dei nostri oggetti e che funziona in modo simile alla proprietà `name`. In effetti, dovrebbe essere usata al posto della proprietà `name` dove esiste un problema di ambiguità. Cambiamo le linee rilevanti per la porta del gabinetto e la chiave del gabinetto:

```
[TYPE]
    Object  porta_del_gabinetto
        with
            pname  'porta' '.x' 'rossa' '.x' 'del' '.x' 'gabinetto' '.x'
                   'bagno',
            short_name [;
            ...
    Object  chiave_del_gabinetto "chiave del gabinetto" benny
        with
            pname 'chiave' '.x' 'del' '.x' 'gabinetto' '.x' 'bagno',
            article "la",
            ...
```

mentre lasciamo senza modifiche il `fuori_dal_gabinetto`:

```
Object  fuori_dal_gabinetto "gabinetto" bar
    with
        name   'gabinetto'  'cesso'  'toilette'  'toilet'  'ritirata'
               'bagno',
        before [;
        ...
```

Stiamo usando un nuovo operatore – `'.x'` – nella nostra lista di parole `pname`

> Il file di testo `pname.txt` spiega che la prima parola di dizionario alla destra dell'operatore `'.x'` è interpretata come opzionale.

e questo rende le parole `'gabinetto'` e `'bagno'` di minore importanza per tali oggetti, in modo che durante il gioco il giocatore possa riferirsi alla PORTA o alla PORTA DEL GABINETTO, o alla CHIAVE o alla CHIAVE DEL GABINETTO - ma non semplicemente al GABINETTO - quando si riferisce alla porta o alla chiave. E, visto che abbiamo lasciato immutata la proprietà `name` dell'oggetto `fuori_dal_gabinetto` – dove c'è un'altra parola `'gabinetto'` - la proprietà `pname` dirà all'interprete di trascurare la chiave e la porta come possibili oggetti quando il giocatore si riferisce solo al GABINETTO. Vedendola in termini presi da quelli della grammatica italiana, abbiamo detto che "GABINETTO" è un aggettivo nelle frasi "PORTA DEL GABINETTO" e "CHIAVE DEL GABINETTO", ma un sostantivo quando viene usato per riferirsi alla stanza.

Il pacchetto `pname.h` ha delle funzionalità aggiuntive che permettono di gestire frasi più complesse, ma non ne abbiamo bisogno nel nostro gioco d'esempio. Sentitevi liberi, comunque, di leggere il file `pname.txt` e scoprire cosa può fare per voi questa notevole estensione della libreria: è una facile risposta per molti mal di testa da ambiguità.

Non sparate sul barista

Molta azione, nel gioco, ruota attorno a Benny, e la sua definizione merita un po' di attenzione. Spieghiamo cosa vogliamo che accada.

> Allora, la porta è chiusa e il giocatore, dopo aver scoperto quel che dice la nota affissa sulla porta del gabinetto, vorrà infine chiedere a Benny la chiave. Purtroppo Benny permette l'uso del gabinetto solo ai clienti, un commento che farà indicando il menù alle sue spalle. Il giocatore dovrà chiedere un caffè prima, in questo modo qualificandosi agli occhi di Benny come un cliente, e quindi persona autorizzata ad usare il gabinetto. Finalmente! Il giocatore potrà fiondarsi dentro il gabinetto, indossare il costume di Capitan Fato e volare via a risolvere la situazione!

Peccato che il giocatore non ha pagato il caffè, né restituito la chiave del bagno. In queste circostanze Benny dovrà impedire al giocatore di lasciare il locale. Per prevenire inutili complicazioni, ci sarà una moneta vicino al wc, abbastanza per pagare il caffè. E così tutto è risolto; abbastanza semplice da descrivere - non così semplice da programmare. Ricordate la definizione base di Benny dal capitolo precedente:

```
Object  benny "Benny" bar
    with   name 'benny',
           description
               "Un uomo ingannevolmente GRASSO dotato di incredibile
                agilità, Benny intrattiene i clienti schiacciandosi
                noci di cocco sulla fronte quando è dell'umore
                giusto.",
    has scenery animate male proper transparent;
```

Ora possiamo aggiungere un po' di complessità, iniziando dalla proprietà `life`. In forma generica:

```
life [;
    Give: … codice per dare oggetti a Benny
    Attack: … codice per gestire le mosse aggressive del giocatore
    Kiss: … codice per il giocatore che diventa affettuoso con Benny
    Ask,Tell,Answer: … codice per gestire le conversazioni
 ],
```

Abbiamo visto qualcuna di queste azioni prima. Prendiamoci cura delle più facili:

```
[TYPE]
    Attack:
        if (costume has worn) {
            deadflag = 4;
            print "Davanti agli occhi pieni di orrore della gente
                circostante, salti MAGNIFICIENTEMENTE OLTRE il bancone e
                attacchi Benny con RIMARCHEVOLE, anche se NON
                sufficiente velocità. Benny ti riceve con uno sleale
                uppercut che spedisce la tua MASCELLA DI GRANITO
                attraverso tutto il locale.^^ ~Questi uomini in pigiama
                pensano di potersela prendere con gente innocente,~
                sbuffa Benny, mentre la SPETTRALE mano dell'OSCURITÀ
                cala sulla tua vista e tu perdi conoscenza.";
        }
```

```
        else
            "Questo non è un atto che potrebbe compiere il MITE John
             Covarth.";
    Kiss:  "Non c'è tempo per INSENSATE infatuazioni.";
    Ask,Tell,Answer:
            "Benny è troppo occupato per mettersi a chiacchierare.";
```

Attaccare Benny non è una mossa saggia. Se il giocatore è sempre vestito come John Covarth, il gioco mostra un messaggio che rifiuta l'atto di violenza per continuare ad interpretare un inutile fallito. Invece, se Capitan Fato tenta l'azione, scopriremo che Benny ha delle doti nascoste, ed il gioco sarà perso. Baciare e conversare sono proibite da un paio di messaggi appropriati.

L'azione `Give` (dare) è un po' più complicata, visto che Benny reagisce a certi oggetti in modo speciale. Tenete a mente che la definizione di Benny deve memorizzare se il giocatore ha chiesto un caffè (diventando quindi un cliente e perciò meritevole della chiave), se il caffè è stato pagato, e se la chiave del gabinetto è stata restituita. La soluzione, ancora una volta (questa è proprio una funzionalità utile), è quella di usare ulteriori proprietà locali:

```
[TYPE]
    Object  benny "Benny" bar
        with  name 'benny',
            description
                 "Un uomo ingannevolmente GRASSO dotato di incredibile
                  agilità, Benny intrattiene i clienti schiacciandosi
                  noci di cocco sulla fronte quando è dell'umore
                  giusto.",
            chiesto_caffe    false,  ! il giocare ha chiesto un caffè?
            caffe_non_pagato false,  ! Benny aspetta di essere pagato?
            chiave_non_resa  false,  ! Benny aspetta la chiave indietro?
            life [;
            ...
```

Ora siamo pronti per sistemare l'azione `Give` della proprietà `Life`, che gestisce i comandi come DAI LA CHIAVE A BENNY (tra poco arriveremo all'azione `Give` della proprietà `Orders`, che gestisce comandi come BENNY, DAMMI LA CHIAVE):

```
[TYPE]
    Give:
        switch (noun) {
            vestiti:
                "Hai BISOGNO degli anonimi vestiti di John Covarth.";
            costume:
                "Hai BISOGNO della tua stupenda tuta ANTI-ACIDO.";
            chiave_del_gabinetto:
                self.chiave_non_resa = false;
                move chiave_del_gabinetto to benny;
                "Benny annuisce mentre tu MIRABILMENTE gli rendi la
                 chiave.";
            moneta:
                remove moneta;
                self.caffe_non_pagato = false;
                print  "Con meravigliose movenze da ILLUSIONISTA, fai
                        apparire una moneta dal tuo ";
                if (costume has worn)
                       print "costume a PROVA DI PROIETTILE";
                else print "normale vestito di tutti i giorni";
```

```
            " come se fosse uscita fuori dall'orecchio di Benny! Le
             persone attorno a te applaudono educatamente. Benny
             prende la moneta, e la morde SOSPETTOSO. ~Grazie,
             signore. Torni quando vuole,~ dice.";}
```

L'azione `Give` della proprietà `Life` tiene nella variabile `noun` l'oggetto offerto all'NPC. Ricordate che potete usare l'istruzione `switch` come abbreviazione per:

```
if (noun == costume) { qualcosa };
if (noun == vestiti) { qualcosa };
...
```

Non permetteremo al giocatore di dar via i suoi vestiti o il suo costume (un'azione improbabile, ma non si sa mai). La chiave del gabinetto e la moneta sono trasferiti con successo. Alla proprietà `chiave_non_resa` assegneremo il valore `true` quando riceveremo la chiave del gabinetto da Benny (non abbiamo ancora scritto il codice per questo), e ora, quando ridiamo la chiave indietro, le assegnamo di nuovo il valore `false`. L'istruzione `move` serve a trasferire l'oggetto dall'inventario del giocatore a Benny, e finalmente mostriamo un messaggio di conferma. Con la moneta troviamo una nuova istruzione, `remove`. Questa toglie l'oggetto dall'albero degli oggetti, in modo che non abbia genitore. L'effetto è quello di farlo scomparire dal gioco (anche se non state distruggendo l'oggetto permanentemente - e infatti potete riportarlo nell'albero degli oggetti con un'istruzione `move`); per quanto riguarda il giocatore, non c'è un'altra MONETA da trovare. Alla proprietà `caffe_non_pagato` assegneremo il valore `true` quando Benny ci serve una tazza di caffè (lo vedremo tra poco); ora la riportiamo a `false`, il che libera il giocatore dal suo debito. Il tutto termina con l'istruzione `"..."`, stampa e ritorna, che dice al giocatore che l'azione ha avuto successo.

Perché abbiamo spostato la chiave ridandola a Benny, ma abbiamo rimosso la moneta? Una volta che il giocatore si è qualificato come cliente ordinando un caffè, potrà chiedere e restituire la chiave quante volte vuole, così sembra importante tenere la chiave a disposizione. La moneta, invece, serve una volta sola. Non permetteremo al giocatore di ordinare più di un caffè, in modo da impedire che il suo debito possa crescere all'infinito - inoltre è venuto qui a cambiarsi, non a sorbire bevande a base di caffeina. Una volta che la moneta viene usata per pagare, essa sparisce, probabilmente nelle avide tasche di Benny. Non c'è motivo di preoccuparsene ulteriormente.

L'oggetto `benny` ha bisogno anche di una proprietà `orders`, che si occupi delle richieste del giocatore per un caffè e per la chiave, e per respingere altre richieste. L'azione `Give` in una proprietà `orders` si occupa proprio di comandi come CHIEDI LA CHIAVE A BENNY e BENNY, DAMMI LA CHIAVE. La sintassi è simile a quella della proprietà `life`:

```
orders [;
    ! gestisce CHIEDI LA CHIAVE A BENNY e BENNY, DAMMI LA CHIAVE
    Give:
        if (second ~= player or nothing)"Benny ti guarda stranito.";
        switch (noun) {
            chiave_del_gabinetto:
                if (chiave_del_gabinetto in player)
                          "Ma tu HAI già la chiave.";
                if (self.chiesto_caffe == true) {
                    if (chiave_del_gabinetto in self) {
                        move chiave_del_gabinetto to player;
                        self.chiave_non_resa = true;
                        "Benny getta la chiave delle toilettes sul
                         bancone, da cui tu la prendi con un destro e
                         preciso movimento della tua mano SUPER-
                         AGILE.";
                    }
                    else
                        "~L'ultimo posto in cui ho visto quella
                         chiave, è stata la TUA mano, ~ mugugna Benny.
                         ~Assicurati di restituirgliela prima di andar
                         via.~";
                }
                else
                    "~Il gabinetto è solo per i clienti.~ mormora,
                     indicando con il dito il menu alle sue spalle.";
            caffe:
                if (self.chiesto_caffe == true)
                    "Un caffè mi sembra abbastamza.";
                    move caffe to bancone;
                    self.chiesto_caffe = true;
                    self.caffe_non_pagato = true;
                    "In sole due mosse aggraziate, Benny posa di
                     fronte a te il suo famosissimo Caffè
                     macchiato.";
            cibo:
                "Mangiare ti prenderebbe troppo tempo, devi cambiarti
                 ORA.";
            menu:
                "Con solo un piccolissimo singhiozzo, Benny fa un
                 cenno verso il menu affisso al muro alle sue
                 spalle.";
            default:
                "~Non credo sia sul menù, signore.~";
        }
    ],
```

- Controlliamo il valore di `second` per intercettare gesti ultra generosi come BENNY, DAI IL CAFFE AGLI AVVENTORI. Quindi consideriamo le richieste potenziali.
- **Chiave del gabinetto**: anzitutto controlliamo se il giocatore ha già la chiave oppure no, e nel caso mostriamo un messaggio e interrompiamo l'esecuzione grazie al `return true` implicito dell'istruzione `"..."`. Se il giocatore non ha la chiave, controlliamo se ha già chiesto un caffè, valutando la proprietà `chiesto_caffe`. Se ciò è vero, dobbiamo controllare anche che la chiave sia in possesso di Benny – un giocatore perverso potrebbe prendere la chiave, lasciarla da qualche parte e quindi richiederla

nuovamente a Benny: se ciò dovesse accadere visualizziamo un messaggio con cui precisiamo che nessuno può ingannare Benny. Se entrambe le condizioni sono vere allora il giocatore ottiene la chiave, e questo vuol dire che dovrà restituirla – la proprietà `chiave_non_resa` prende il valore `true` – e mostriamo un messaggio adatto. Se invece il giocatore non ha chiesto un caffè, Benny rifiuta di obbedire e di consegnare la chiave, menzionando per la prima volta il menù dove il giocatore potrà vedere il disegno di una tazza di caffè quando lo ESAMINA. Fate attenzione a notare come tutte le clausole `else` appaiano con l'appropriata istruzione `if`, fornendo responsi per ogni condizione che non è stata incontrata con successo.

- **Caffè**: controlliamo se il giocatore ne ha già chiesto uno, valutando la proprietà `chiesto_caffe`, e rifiutiamo di servirne un altro se fosse vero. Se la proprietà ha valore `false`, piazziamo un caffè sul bancone e assegniamo true alle proprietà `chiesto_caffe` e `caffe_non_pagato`. La parte del messaggio la conoscete già.
- **Cibo**: introdurremo un oggetto per gestire tutte quelle cose commestibili e deliziose che possiamo trovare nel bar, specialmente quelle (come i `'pezzi dolci'` e i `'sandwich'`) menzionati nelle nostre descrizioni. Anche se questo oggetto non è ancora stato definito, introduciamo anche questo caso per reagire alla gola del giocatore se provasse a chiedere a Benny del cibo.
- **Menù**: la nostra risposta di default "Non credo sia sul menù, signore" non è molto appropriata se il giocatore chiede un menù, così ne forniamo una migliore.
- **Default**: questo si occupa di ogni altra cosa che il giocatore può chiedere a Benny, visualizzando la sua risposta.

E prima che ve ne siate potuti rendere conto, l'oggetto Benny è finito; però non festeggiate subito. Ci sono ancora dei comportamenti di Benny che, curiosamente, non sono codificati all'interno dell'oggetto Benny; stiamo parlando di come reagisce Benny se il giocatore prova ad andarsene senza pagare o senza restituire la chiave. Vi abbiamo promesso che Benny avrebbe fermato il giocatore, e così sarà. Ma dove?

Dobbiamo rivisitare l'oggetto `bar`:

```
[TYPE]
    Room bar "Il bar di Benny"
        with
            description [;
                print "Benny offre la MIGLIORE selezione di pezzi dolci e
                     sandwich. I clienti riempiono il bancone dove Benny
                     in persona riesce a servire, cucinare e riscuotere
                     senza la minima esistazione. Sulla parete nord del
                     bar vedi una porta rossa che conduce al gabinetto.";
            ],
            appenauscito false,  ! Prima apparizione di Capitan Fato?
            before [;
                Go:
                !il giocatore se ne sta per andare è diretto in strada?
```

```
            if (noun ~= s_obj) return false;
            if(benny.caffe_non_pagato==true ||
                benny.chiave_non_resa==$true) {
                print "Come accenni ad uscire per strada, la grossa
                   mano di Benny si appoggia sulla tua spalla.";
                if (benny.caffe_non_pagato == true &&
                    benny.chiave_non_resa == true)
                        "^^~Hey! Hai ancora la mia chiave e non hai
                         pagato il caffè. Ti sembro forse uno
                         stupido?~ Ti scusi come soltanto un EROE sa
                         fare e torni all'interno.";
                if (benny.caffe_non_pagato == true)
                        "^^~Aspetta un minuto, Amico,~ dice. ~Stiamo
                         cercando di sgattaiolare via senza pagare,
                         vero?~ Mormori velocemente una scusa e torni
                         all'interno del bar. Benny torna ai suoi
                         compiti continuando a guardarti sospettoso.";
                if (benny.chiave_non_resa == true)
                        "^^~Dove credi di andare con la chiave del
                         gabinetto?~ dice. ~Sei forse un ladro?~
                         Mentre Benny ti spinge di nuovo all'interno
                         del locale, lo rassicuri velocemente
                         spiegando che si è trattato solo di uno
                         STUPEFACENTE errore.";
            }
            if (costume has worn) {
              deadflag = 5;                ! hai vinto!
              "Esci in strada, dove le persone di passaggio
               riconoscono la STRAVAGANZA arcobaleno del costume di
               Capitan FATO e gridano il tuo nome con stupore mentre
               tu SALTI con forza sensazionale nel cielo BLU del
               mattino!";
            }
        ],
        after [;
            Go: ! Il giocatore è appena arrivato. Viene dal bagno?
            if (noun ~= s_obj) return false;
            if (costume has worn && self.appenauscito == false) {
                self.appenauscito = true;
                StartDaemon(clienti);
            }
        ],
        s_to strada,
        n_to porta_del_gabinetto;
```

Ancora una volta notiamo che la soluzione ad un problema non è necessariamente unica. Ricordate cosa abbiamo visto quando ci occupavamo della descrizione del giocatore: avremmo potuto assegnare un nuovo valore alla variabile `player.description`, ma abbiamo scelto di usare l'oggetto `LibraryMessages`. Questo è un caso simile. Il codice che permette a Benny di intercettare il giocatore distratto, forse, avrebbe potuto essere aggiunto ad una proprietà `daemon` nella definizione di Benny. Però, visto che l'azione che vogliamo intercettare è sempre la stessa, ed è un'azione di movimento (quando il giocatore prova ad uscire dal bar), è possibile scrivere il suo codice intercettando l'azione `Go` dell'oggetto `bar`. Entrambe le soluzioni sarebbero state giuste, ma questa è un po' più semplice.

Abbiamo aggiunto una proprietà `before` all'oggetto `bar` (una un po' lunghetta), che si occupa solo dell'azione `Go`. Questa tecnica permette d'intercettare il giocatore che prova ad uscire da una stanza prima che il movimento abbia luogo, un buon momento per interferire se vogliamo impedire la fuga. La prima riga:

```
if (noun ~= s_obj) return false;
```

dice all'interprete che vogliamo occuparci solo dei movimenti che conducono verso sud, lasciando che l'interprete applichi le normali regole per le altre direzioni possibili; l'operatore ~= sta per "non uguale a". Da qui in poi sono solo condizioni e altre condizioni. Il giocatore può provare ad uscire:

- senza aver pagato il caffè e senza aver restituito la chiave
- avendo pagato il caffè, ma senza restituire la chiave
- avendo restituito la chiave, ma senza aver pagato il caffè
- pulito da ogni peccato e di nulla colpevole agli occhi degli uomini (beh, agli occhi di Benny, almeno)

Le prime tre sono coperte dal test:

```
if (benny.caffe_non_pagato == true || benny.chiave_non_resa == true)
```

cioè, se il caffè non è stato pagato *o* la chiave non è stata restituita. Quando questa condizione è falsa, vuol dire che entrambi i malcomportamenti sono stati evitati e che il giocatore può uscire liberamente. Se però questa condizione è vera, la mano di Benny cadrà sulla spalla del giocatore e il gioco mostrerà un messaggio diverso, a seconda di quale errore o quali errori il personaggio ha commesso.

Se il giocatore può andare, *e* sta indossando il suo costume da combattente del crimine, il gioco è vinto. Vi diremo come nel prossimo capitolo, dove concluderemo la creazione del gioco.

13. Capitan Fato: ultimo atto

Probabilmente sarete contenti di sapere che l'avventura di Capitan Fato è vicina alla sua conclusione. Ci sono ancora alcuni oggetti secondari da definire – il gabinetto, i vestiti dell'eroe e naturalmente il suo costume – ma cominciamo innanzitutto dal decorare meglio il nostro bar.

Più guarnizioni culinarie

Non dobbiano dimenticare un paio di piccoli dettagli nella locazione del bar:

```
[TYPE]
    Object  cibo "Gli spuntini di Benny" bar
        with
            Name   'pezzi'  'dolci'  'cibo'  'sandwich'  'pezzo'  'dolce'
                   'pasta' 'paste' 'spuntino' 'spuntini',
            before [; "Adesso non è il momento di pensare al CIBO."; ],
        has scenery proper;

    Object menu "menu" bar
      with name 'menu' 'lista' 'listino',
            description
                "Il menu appeso al muro elenca tutti i cibi e bevande che
                 Benny può servire. Peccato che tu non abbia mai imparato
                 a leggere, ma fortunatamente c'è il disegno di una
                 grossa tazza di caffè fra tutte le altre scritte
                 incomprensibili.",
            before [;
                Take:
                    "Il menu è affisso al muro alle spalle di Benny.
                     Inoltre è inutile SCRITTURA.";
            ]
        has scenery;
```

Ed un oggetto non così semplice:

```
[TYPE]
    Object caffe "tazza di caffè" benny
        with
            name   'tazza' 'di' 'caffe' 'caffè' 'macchiato' 'cappuccino'
                   'capucino' 'cappucino' 'capuccino',
            description [;
                if (self in benny)
                    "Il disegno sul menù ha SICURAMENTE un
                     bell'aspetto.";
                else "Aroma delizioso.";
            ],
            before [;
                Take,Drink,Taste:
                    if (self in benny)
                        "Forse dovresti ordinarne uno a Benny.";
                    else {
                        move self to benny;
                        "Prendi la tazzina e ne bevi un sorso. La
                         REPUTAZIONE MONDIALE di Benny è ben meritata.
                         Appena finisci, Benny porta via la tazzina. ~Il
                         caffè viene un'euro, signore.~";
```

```
                if (benny.caffe_non_pagato == true)
                    "~Viene un euro, Signore~";
                else "" ;
            }
        Buy:
            if (moneta in player) <<Give moneta benny>>;
            else "Non hai soldi.";
        Smell:
            "Se la tua IPERATTIVA ghiandola pituitaria è
             affidabile, è una miscela colombiana.";
    ],
    has female;
```

Non vi è nulla di veramente nuovo in questo oggetto (oltre al fatto che la proprietà `name` tenga conto di alcuni errori di ortografia del giocatore), ma notate come non abbiamo rimosso (`remove`) il caffè dopo che il giocatore lo ha bevuto. Per un apparentemente assurdo capriccio, il caffè ritorna magicamente nelle mani di Benny (sebbene questa non sia una informazione che il giocatore debba sapere). Perché? Vi chiederete... E' presto detto. Dopo che si è rimosso un oggetto dal gioco, se il giocatore tenta di ESAMINARLO, l'interprete dirà in maniera poco appropriata, "Non vedi nulla del genere". Per di più, se il giocatore chiede un secondo caffè a Benny, una volta che il primo sia rimosso, Benny risponderà "Non credo sia sul menù, signore" - una sfacciata bugia - ossia la risposta di default della proprietà `orders` di Benny. Dal momento che l'oggetto caffè rimosso non appartiene più a Benny, non è più un un sostantivo che il giocatore gli può CHIEDERE. Mentre facendolo tornare un oggetto figlio (child) del barista (che ha impostato l'attributo `trasparent`), il caffè rimane ancora un oggetto a cui il giocatore può fare riferimento.

Ci assicureremo, poi, che il giocatore non possa ordinare più tazze di caffè grazie alla proprietà `chiesto_caffe`, che rimane vera dopo la prima volta. Ci assicureremo, inoltre, che Benny non chieda altri soldi al giocatore che ha già pagato, prima stampando il messaggio "Prendi la tazzina…" e quindi testando la proprietà `chiesto_caffe` di Benny. Se è vera allora stamperemo semplicemente la richiesta del denaro con il `return true` incorporato. Se è `false`, il giocatore ha già pagato, e pertanto non c'è niente altro di interessante da dire. Comunque, abbiamo ancora bisogno di terminare il messaggio incompleto con una nuova linea (`newline`), e un `return true` dalla routine della proprietà; *potremmo* usare le istruzioni `{ print "^"; return true; }`, ma una istruzione `""` vuota fa la stessa cosa molto più brevemente.

Gabinetto o spogliatoio?

Propenderemo certamente per la seconda definizione, visto che, al momento, è l'unico posto al riparo da sguardi curiosi dove il nostro eroe potrà trasformarsi da uomo debolucciо in nemesi di ogni essere maligno. E non è *proprio* il caso di farsi troppi scrupoli e di impantanarci nei dettagli sulle reali funzioni di una stanza. Non c'è molto da dire sulla locazione gabinetto e sui suoi contenuti, sebbene vi siano alcuni effetti speciali:

```
[TYPE]
    Room    gabinetto "Un gabinetto unisex"
        with   description
                   "Una stanza quadrata, incredibilmente PULITA, dalle
                    pareti ricoperte di mattonelle di ceramica, che non
                    contiene molto di più di un gabinetto e un
                    interruttore. L'unica uscita è a sud, attraverso la
                    porta che riconduce al bar.",
               s_to porta_del_gabinetto,
        has   ~light scored;

    Appliance cesso "gabinetto" gabinetto
        with
            name   'gabinetto'   'wc'   'cesso'   'water'   'water-closed'
                   'tazza',
            before [;
                Examine:
                    if (moneta in self) {
                        move moneta to parent(self);
                        "L'ultima persona ha CIVILMENTE tirato l'acqua
                         dopo aver usato il gabinetto, ma ha dimenticato
                         di raccogliere la PREZIOSA moneta che è caduta
                         dai suoi pantaloni.";
                    }
                Receive:
                    "Mentre qualsiasi altro MORTALE potrebbe gettare
                     QUALSIASI cosa senza pensarci ", (artin) self, ", ti
                     ricordi i profondi insegnamenti del tuo mentore, il
                     Duca ELEGANTE, di come usare e cavalcare i cessi.";
            ];

    Object  moneta "moneta" gabinetto
        with
            name 'moneta' 'euro' 'soldi',
            description "E' una moneta da un EURO.",
            before [;
                Drop:
                    if (self notin player) return false;
                    "Lasciare una moneta tanto preziosa? Ha, ha! Questa
                     deve essere una dimostrazione del tuo ULTRA-FRIVOLO
                     senso dell'umorismo!";
            ],
            after [;
                Take:
                    "Ti accosci nella posizione del DRAGO DORMIENTE e
                     senza indugio raccogli e intaschi la moneta con la
                     PRESA SOVRANA.";
            ],
        has scored female;
```

Inizialmente abbiamo posto la moneta come oggetto figlio (`child`) del gabinetto (così che si possa facilmente verificare `if(coin in self)`). Dal momento che non abbiamo impostato l'attributo `trasparent` per il gabinetto, la moneta non sarà visibile al giocatore fino a quando non tenterà di ESAMINARE il gabinetto, azione che sposterà la moneta nella locazione gabinetto. Una volta raccolta, la moneta rimarrà nell'inventario fino a quando il giocatore la darà a Benny, visto che abbiamo bloccato l'azione POSA MONETA (`drop`) per aiutare il giocatore a FARE la Cosa Giusta.

L'oggetto gabinetto include una serie di utili sinonimi nella sua proprietà `name`, inclusa la nostra parola preferita `'gabinetto'`. Ciò non costituirà un problema: gli altri oggetti qui presenti che possono avere la parola GABINETTO nel loro nome – ovvero, la chiave e la porta – usano entrambi la propretà `pname` che pone per loro l'uso della parola `'gabinetto'` come aggettivo di importanza secondaria.

Notate che qui sono presenti gli unici due attributi `scored` del gioco. Il giocatore guadagnerà un punto entrando nella locazione gabinetto, e un altro punto trovando e raccogliendo la moneta.

Potreste anche aver notato che abbiamo forzatamente eliminato l'attributo `light`, ereditato dalla classe `Room`. Ci troviamo infatti in una stanza senza finestre e, per aggiungere un tocco di realismo, è il caso di rendere il gabinetto una locazione priva di luce, il che ci permetterà di parlarvi del comportamento di default di Inform nel caso non vi sia una sorgente di luce in una locazione. Comunque, andiamo prima a definire l'interruttore della luce menzionato nella descrizione della stanza per aiutare il giocatore nel suo cambio d'abito.

```
[TYPE]
    Appliance interruttore "interruttore" gabinetto
        with
            name 'interruttore',
            description
                "Un notevole PRODIGIO di tecnologia e SCIENZA, elegante e
                 FACILE da usare.",
            before [;
                Push:
                    if (self has on) <<SwitchOff self>>;
                    else             <<SwitchOn  self>>;
            ],
            after [;
                SwitchOn:
                    give self light;
                    "Accendi la luce del gabinetto.";
                SwitchOff:
                    give self ~light;
                    "Spegni la luce del gabinetto.";
            ],
        has switchable ~on;
```

Per favore, notate l'inserimento dei nuovi attributi `switchable` e `on`.

`switchable` permette all'oggetto di essere acceso o spento, ed è un attributo tipico di lanterne, calcolatori, televisioni, radio, e così via. La libreria automaticamente completa la descrizione di questi oggetti indicando se al momento sono accesi o spenti:

```
> X INTERRUTTORE DELLA LUCE

Un noto PRODOTTO della tecnologia MODERNA, elegante e FACILE da
usare.

L'interruttore della luce è al momento acceso.
```

A questo punto vi sono due nuove azioni pronte per l'uso, `SwitchOn` (accendi) e `SwitchOff` (spegni). Lasciate a se stesse, esse modificano lo stato dell'oggetto

portandolo da ACCESO a SPENTO o viceversa e visualizzando un messaggio del tipo:

```
Hai acceso la lanterna.
```

Esse si preoccupano anche di controllare se il giocatore si sbaglia e prova ad accendere (o spegnere) un oggetto che è già acceso (o spento). Come fa la libreria a sapere lo stato di un oggetto? Ciò avviene grazie all'attributo `on`, che viene impostato o cancellato automaticamente secondo la necessità. Potete, naturalmente, impostarlo o cancellarlo manualmente, come ogni altro attributo, con l'istruzione `give`:

```
give self on;
give self ~on;
```

e controllare se un oggetto `switchable` è acceso o spento con il test:

```
if (interruttore has on)...
if (interruttore hasnt on)...
```

Un oggetto `switchable` di default è SPENTO. Comunque, avrete notato che la linea `has` della definizione dell'oggetto include `~on`:

```
has   switchable ~on;
```

Siete sicuri che questa istruzione stia dicendo "non-acceso"? Siete certi che non sarebbe accaduto lo stesso se la linea non avesse menzionato del tutto l'attributo?

```
has   switchable;
```

Bene, avete perfettamente ragione. Aggiungere l'attributo `~on` non ha alcun effetto sul gioco, ma ciò nonostante è una buona idea. È un promemoria, un modo per ricordarci che iniziamo il gioco con questo attributo disattivato, e che in qualche punto dovremo impostarlo per i nostri scopi. Credeteci: vale la pena approfittare di queste piccole opportunità di aiutare voi stessi.

Andiamo a vedere come funziona il nostro interruttore. Intercettiamo le azioni `SwitchOn` e `SwitchOff` nella proprietà `after` (quando oramai l'accensione o spegnimento dell'interruttore ha fatto effetto) e usiamole per dare *luce* (attributo `light`) all'interruttore.

Uh, aspettate un attimo. Luce all'interruttore? Perché non invece alla locazione gabinetto? Bene, c'è una ragione e la vedremo fra un minuto. Per adesso, ricordate solo che, per far sì che il giocatore possa vedere ciò che lo circonda, basta che vi sia un oggetto nella locazione con l'attributo `light` impostato. Non c'è bisogno che sia la locazione stessa (anche se normalmente è conveniente che lo sia).

Dopo avere impostato l'attributo `light`, visualizziamo un messaggio sullo schermo, per evitare quello di default:

```
Hai acceso l'interrutore.
```

che non è proprio elegante. Il giocatore potrebbe provare anche a PREMERE l'INTERRUTTORE, così abbiamo intercettato questo tentativo nella proprietà `before` e lo abbiamo ridirezionato verso le azioni `SwitchOn` e `SwitchOff`, controllando prima che sia necessario testando l'attributo `on`. Infine, abbiamo reso l'interruttore un membro della classe `Appliance`, così il giocatore non potrà portarselo via.

> NOTA: ricordate cosa abbiamo detto a proposito dell'ereditarietà delle classi? Non importa cosa definite nella classe, la definizione dell'oggetto ha la priorità. La classe `Appliance` definisce una risposta per l'azione `Push` (premi), ma noi la sovrascriviamo qui con un nuovo comportamento.

E luce fu

Il giocatore entra nel gabinetto e

```
Oscurità
E' completamente buio, non riesci a vedere niente.
```

Oops! nessuna descrizione del gabinetto, nessun riferimento all'interruttore della luce, niente. È però ragionevole pensare che se abbiamo aperto la porta del gabinetto per accedervi, un po' di luce del bar filtri attraverso l'uscio illuminando il bagno - almeno fino a quando il giocatore non deciderà di chiudere la porta del gabinetto. Così potrebbe essere una buona idea aggiungere qualche linea di codice all'oggetto porta per tenere conto di questa situazione. Un paio di istruzioni nella proprietà `after` saranno sufficienti:

```
[TYPE]
    after [ ks;
        Unlock:
            if (self has locked) return false;
            print "Apri ", (the) self, ".^";
            ks = keep_silent; keep_silent = true;
            <Open self>; keep_silent = ks;
            return true;
        Open:  give gabinetto light;
        Close: give gabinetto ~light;
         ],
```

E questa è la ragione per cui l'interruttore della luce non imposta l'attributo `light` sulla locazione gabinetto, ma lo fa su se stesso. Avremmo qualche difficoltà se permettessimo allo stato aperto/chiuso della porta di controllare la luce dell'oggetto locazione gabinetto, e allo stato acceso/spento dell'interruttore la luce dell'interruttore stesso. Così è meglio avere un unico interruttore per la luce che fa tutto. Fortunatamente, il giocatore non si accorgerà mai di questo artifizio.

> NOTA: potrebbe accorgersene in qualche modo? Beh, se il giocatore potesse PRENDERE l'interruttore della luce (cosa che abbiamo evitato), dando il comando INVENTARIO, il trucco verrebbe svelato, visto che ogni sorgente di luce viene contrassegnata dalla libreria con la parola (acceso).

Il giocatore entra nel gabinetto e

```
Un gabinetto unisex.
Una stanza quadrata, incredibilmente PULITA, dalle pareti
ricoperte di mattonelle di ceramica, che non contiene molto di più
di un gabinetto e un interruttore. L'unica uscita è a sud,
attraverso la porta che riconduce al bar.

[Il tuo punteggio è appena aumentato di un punto.]
```

Meglio. Ora, supponete che il giocatore chiuda la porta.

```
>CHIUDI PORTA
Ora hai chiuso la porta che conduce al bar.

È completamente buio qui!

>L

Oscurità
È completamente buio, e non riesci a vedere niente.
```

Sì, nessun problema. Abbiamo già detto che c'è un interruttore della luce. Sicuramente il giocatore ora proverà:

```
>ACCENDI L'INTERRUTTORE
Non vedi nulla del genere.
```

Oops! Le cose si fanno complicate al buio. Probabilmente è il momento di lasciare questo posto e provare un altro approccio:

```
>APRI LA PORTA
Non vedi nulla del genere.
```

E ciò mostra una di quelle cose terribili che accadono quando in un gioco si cade in una locazione buia. Non potete vedere nulla e potete fare veramente poco in questa situazione. Tutti gli oggetti, con l'eccezione di quelli presenti nel vostro inventario sono fuori portata, irraggiungibili, come se non esistessero. Peggio ancora, se POSATE un oggetto tra quelli che state portando, esso sarà irrimediabilmente perso nel buio, introvabile fin quando non tornerà la luce.

Il giocatore che senza dubbio è immerso nella fantasia del gioco, sarà ora un po' interdetto. "Sono in un piccolo bagno, e non posso raggiungere la porta che ho appena chiuso?" Il giocatore ha ragione, naturalmente[15].

Le locazioni buie sono un classico dei giochi tradizionali. Solitamente ci si sposta in una direzione mentre si sta cercando un tesoro in una qualche sorta di caverna, e improvvisamente si arriva in un antro oscuro. È considerato un buon comportamento da parte dell'avventura disabilitare l'esplorazione di un

[15] Stiamo alludendo qui al classico concetto di *mimesi*. In un articolo scritto nel 1996, Roger Giner-Sorolla ha scritto: "Considero la narrativa ben fatta un'imitazione, o una "mimesi", della realtà, sia che si svolga in questo mondo che in un altro. La buona narrativa permette al lettore di entrare, temporaneamente, nella realtà di quel mondo e di crederci. Un crimine contro la mimesi è ogni aspetto di un gioco di IF che spezza la coerenza di questo mondo fittizio, di questa appresentazione di una realtà." Il testo completo dell'articolo tradotto in italiano può essere trovato su http://www.avventuretestuali.com.

territorio buio e sconosciuto, ed è una convenzione sbarrare il passaggio al giocatore fino a quando non torna con una sorgente di luce. Comunque, se lo scenario del gioco prevede, supponiamo, la casa del personaggio giocatore, un piccolo appartamento di due stanze, e non c'è luce nella cucina, dovremmo aspettarci che il proprietario della casa sappia muoversi lo stesso al suo interno, forse anche solo per raggiungere un interruttore o che riesca a raggiungere il frigorifero anche al buio.

In questo caso siamo in una situazione simile. La logica del gioco richiede che un giocatore al buio dovrebbe essere in grado di aprire la porta e probabilmente di accendere un interruttore della luce che ha appena incontrato. Abbiamo detto che un oggetto è a portata (scope) quando è nella medesima locazione del giocatore. L'oscurità cambia questa regola. Tutti gli oggetti non trasportati direttamente dal giocatore sono fuori portata (out of scope).

Uno dei vantaggi di un sistema di programmazione avanzato come Inform è la flessibilità di cambiare tutti i comportamenti di default per venire incontro alle esigenze particolari. Anche qui il problema non è differente. C'è una serie di routine e funzioni atte a metter mano su cosa sia a portata (scope) del giocatore e quando. Vedremo solo un piccolo esempio che ci aiuterà a risolvere il nostro problema. Nella sezione `"entry point routines"` del nostro gioco -dopo la routine `Initialise`, per esempio- includeremo le seguenti linee di codice:

```
[ InScope person;
    if (person == player && location == thedark &&
              real_location == gabinetto) {
        PlaceInScope(interruttore);
        PlaceInScope(porta_del_gabinetto);
    }
    return false;
];
```

`InScope(`*`oggetto_attore_id`*`)` è una routine punto d'ingresso (`entry point`) che permette d'intervenire sulle regole che mettono alla portata dell'`oggetto_attore_id` (personaggio giocatore o un qualsiasi personaggio non giocatore) gli altri oggetti. La definiamo con una variabile (il cui nome può essere scelto a piacere, ma è una buona idea scegliere un nome che ci ricordi cosa rappresenta la variabile), `person`, e quindi eseguiamo una verifica per vedere se il giocatore è contemporaneamente nel gabinetto ed al buio.

Vi abbiamo già accennato al fatto che la variabile `location` contiene il valore corrente della locazione in cui il giocatore si trova. Quando, però, non c'è luce, la variabile `location` assume il valore dello speciale oggetto della libreria chiamato `thedark` (oscurità). Non importa se abbiamo 10 stanze buie nel nostro gioco; `location` assumerà in tutti i casi il valore `thedark`. C'è un'altra variabile, chiamata `real_location`, che contiene il valore della locazione in cui è il giocatore *anche quando non vi sia luce per vedere.*

Pertanto la verifica:

```
if (person == player && location == thedark && real_location ==
gabinetto) ....
```

sta controllando: se l'attore specificato è il personaggo giocatore *e* se si trova al buio *e* se al momento sia per caso nel gabinetto...

In seguito richiamiamo una delle routine della libreria, `PlaceInScope(oggetto_id)` che possiede un nome piuttosto chiaro: essa pone alla portata del giocatore l'oggetto specificato. Nel nostro caso, vogliamo che la porta e l'interruttore della luce siano entrambi raggiungibili dal giocatore, ecco spiegate le altre due linee di codice.

Infine, dobbiamo restituire il valore `false`, poiché abbiamo bisogno che le normali regole sulla portata degli oggetti si applichino per l'attore definito - il giocatore - al resto degli oggetti del gioco (se avessimo restituito il valore `true`, il giocatore avrebbe scoperto che poteva interagire con veramente poco). Ora abbiamo un risultato più logico e piacevole:

```
Oscurità
È completamente buio, e non riesci a vedere niente.

>ACCENDI L'INTERRUTTORE
Accendi la luce del bagno.

Un bagno unisex
Una stanza quadrata, incredibilmente PULITA, dalle pareti ricoperte
di mattonelle di ceramica, che non contiene molto di più di un
gabinetto e un interruttore. L'unica uscita è a sud, attraverso la
porta che riconduce al bar.
```

E lo stesso accadrà per la porta. Notate come la descrizione della locazione venga visualizzata dopo il passaggio dal buio alla luce; questo è il comportamento normale della libreria.

C'è ancora un ultimo problema che, ammettiamolo, potrebbe originarsi da una improbabile, ma non impossibile, serie di azioni. Supponiamo che il giocatore entri nella locazione, chiuda la porta a chiave – il che è possibile al buio ora che abbiamo messo la porta alla portata (scope) del giocatore – e che quindi posi la chiave. Non c'è modo di uscire dal gabinetto e la chiave è scomparsa, inghiottita dall'oscurità – se il giocatore non pensa alla possibilità di accendere l'interruttore della luce, facendo tornare la chiave alla sua portata.

Perché non aggiungere un `PlaceInScope(chiave_del_gabinetto)` alle precedenti routine? Beh, per cominciare, la chiave può essere mossa e trasportata (a differenza della porta e dell'interruttore della luce, che sono invece oggetti fissi al loro posto nella locazione del gabinetto). Supponiamo che il giocatore apra la porta del gabinetto, ma posi la chiave all'interno del bar, quindi entri nel gabinetto e chiuda la porta. La condizione viene soddisfatta e la chiave viene posta alla portata del giocatore,anche se era in un'altra locazione. Come secondo motivo, questo è un gioco semplice con solo pochi oggetti, quindi si può definire una regola per ognuno di essi; ma in quasiasi gioco più corposo, potreste trovarvi alle prese con una notevole quantità di oggetti e con la

necessità di creare una regola generale che si applichi ad ognuno (o ad alcuni) di essi.

Abbiamo pertanto bisogno di aggiungere del codice nella routine `InScope` per dire al gioco di mettere alla portata del giocatore tutti gli oggetti che egli posa nel buio, così che si possano recuperare (magari mettendosi a quattro zampe e andando a tentoni). Non vogliamo che il giocatore abbia altri oggetti alla sua portata (come la moneta, per esempio), così potrebbe essere una buona idea avere un modo di verificare se l'oggetto è stato manipolato o trasportato dal giocatore. L'attributo `moved` si adatta perfettamente a questo scopo. La libreria lo imposta per ogni oggetto che il giocatore ha raccolto almeno una volta durante il gioco; oggetti con l'attributo `scenery` o `static`, o quelli che non sono ancora stati visti non possiedono l'attributo `moved`. Ecco qui la rielaborazione della routine `InScope`. Ci sono un paio di nuovi concetti da approfondire:

```
[TYPE]
    [ InScope person item;
       if (person == player && location == thedark && real_location ==
       gabinetto) {
           PlaceInScope(interruttore);
           PlaceInScope(porta_del_gabinetto);
       }
       if (person == player && location == thedark)
           objectloop (item in parent(player))
               if (item has moved) PlaceInScope(item);
       return false;
    ];
```

Abbiamo aggiunto una seconda variabile locale, `item` – ancora una volta, questa è una variabile che abbiamo creato e nominato per conto nostro; non è parte della libreria. Facciamo ora una nuova verifica: se l'attore è il giocatore e la locazione è una qualsiasi locazione buia, esegui una determinata azione. Non abbiamo bisogno di specificare la locazione gabinetto, visto che vogliamo che questa regola si applichi a tutte le locazioni buie (probabilmente vi sarete accorti che il gabinetto *è* l'unica locazione buia del gioco, ma stiamo cercando di usare regole generali).

```
objectloop (variabile) istruzione;
```

è un'istruzione ciclica, una delle quattro definite in Inform. Un ciclo è un costrutto che permette d'eseguire diverse volte un'istruzione (o un blocco di istruzioni). *`objectloop`* esegue *`l'istruzione`* una volta per ogni oggetto definito nella `(variabile)`. Se dovessimo codificarlo:

```
objectloop (item) istruzione;
```

quindi *`l'istruzione`* verrebbe eseguita una volta per ogni oggetto nel gioco. Naturalmente, noi vogliamo eseguire l'istruzione solo per quegli oggetti il cui oggetto genitore (`parent`) è lo stesso dell'oggetto genitore del giocatore: ovvero, per gli oggetti nella stessa locazione del giocatore, quindi il codice sarà:

```
objectloop (item in parent(player)) istruzione;
```

Quali sono le *istruzioni* che vogliamo vedere eseguite ripetutamente?

```
if (item has moved)
    PlaceInScope(item);
```

La verifica: `if (item has moved)` assicura che `PlaceInScope(item)` si relazioni solo con gli oggetti che possiedono l'attributo `moved`. Così: se il giocatore è al buio, la condizione è verificata per ogni oggetto che è nella medesima locazione, uno alla volta. Per ognuno di essi, controlla se è un oggetto che è stato trasportato dal giocatore, nel qual caso, lo pone alla sua portata. Tutti gli oggetti posati all'interno della locazione sono stati trasportati almeno una volta, così lasceremo al giocatore la possibilità di recuperarli anche se non può vederli.

Come vedete, l'oscurità è in parte rognosa da implementare. Se pensate di utilizzare delle locazioni buie nel vostro gioco, tenete a mente che dovrete cimentarvi con un codice abbastanza elaborato (a meno che non manteniate le regole della libreria, nel qual caso il giocatore non potrà fare un granché).

Un fantastico costumino con effetti speciali

Pensiamo come prima cosa ai nostri vestiti, che richiedono alcune azioni fatte su misura. Ecco gli indumenti indossati normalmente da John Covarth:

```
[TYPE]
    Object  vestiti "i tuoi vestiti"
        with
            name 'vestiti' 'vestito' 'ordinari' 'abiti' 'abito',
            description
                "Vestiti perfettamente ORDINARI per un NESSUNO come John
                 Covarth.",
            before [;
                Wear:
                    if (self has worn)
                            "Sei già vestito come John Covarth.";
                    else    "La città ha BISOGNO dei poteri di capitan
                             FATO, non dell'anonimato di John Covarth.";
                Disrobe,Change:
                    if (self hasnt worn)
                        "La tua vista ACUTA ti dice non stai più
                         indossando", (the) self, ".";
                    switch (location) {
                        strada:
                            if (player in cabina)
                                "Non avendo la super-velocità di Superman,
                                 realizzi che sarebbe sconveniente
                                 cambiarti sotto gli occhi delle persone
                                 che passano.";
                            else
                                "In mezzo alla strada? Questo sarebbe uno
                                 SCANDALO, e inoltre rivelerebbe la tua
                                 identità segreta.";
                        bar:
                            "Benny non sopporta le buffonate nel suo
                             locale.";
                        gabinetto:
                            if (porta_del_gabinetto has open)
                                "La porta rimane aperta, e ci sono decine
                                 di occhi curiosi. Dovresti arrestarti da
                                 solo per condotta IMMORALE.";
                            print "Ti togli rapidamente i vestiti e li
                                raccogni in un pacco ULTRA-MINUSCOLO
                                facilmente trasportabile. ";
                            if (porta_del_gabinetto has locked) {
                                give vestiti ~worn; give costume worn;
                                "Poi spieghi il tuo costume in COTONE
                                 INVULNERABILE e ti trasformi in Capitan
                                 FATO, difensore della libertà e
                                 avversario della tirannia!";
                            }
                            else {
                                deadflag = 3;
                                "Stai per infilarti il costume di Capitan
                                 FATO,     quando la porta si apre e una
                                 giovane donna entra. LEI ti guarda e
                                 inizia ad urlare, ~UNO STUPRATORE. UNO
                                 STUPRATORE NUDO NEL BAGNO!!!~^^ Tutti
                                 coloro che erano nel bar giungono in
                                 soccorso, solo per vederti saltare in
                                 modo ridicolo su una gamba sola mentre
```

```
                        cerchi di vestirti. Le loro risate
                        segnano la RAPIDA FINE della tua carriera
                        di combattente del crimine!";
                    }
                thedark:
                    "L'ultima volta che ti sei cambiato al buio,
                     hai indossato il costume al contrario!";
                default:  ! questo caso non dovrebbe mai capitare
                    "Ci sono posti migliori dove cambiarsi
                     d'abito.";
            }
    ],
has clothing proper pluralname;
```

Osservate come l'oggetto si relazioni solo con le azioni `Disrobe` (togli), `Change` (cambia) e `wear` (indossa). `Disrobe` e `wear` sono azioni standard della libreria, già definite in Inform, mentre invece dovremo creare un nuovo verbo per permettere il comando CAMBIA I VESTITI. In questo gioco, `Disrobe` e `Change` sono considerati sinonimi in ogni caso.

L'obiettivo del giocatore è cambiarsi i vestiti, così potremmo aspettarci che egli provi a farlo un po' ovunque. Il compito dell'istruzione `switch` è di offrire risposte diverse a seconda del valore della variabile `location`. Se l'azione viene comandata nella strada (dentro o fuori della cabina) o nel bar allora viene visualizzato un qualche messaggio di rifiuto, solo quando il giocatore riuscirà ad entrare nel gabinetto, e si sarà chiuso dentro a chiave, riuscirà a cambiarsi d'abito con successo. Se la porta è chiusa, ma non a chiave, il giocatore viene interrotto mentre è mezzo nudo da una donna nervosa che comincia a bussare, e la partita è persa (la situazione per il giocatore non è così tragica come sembra, infatti egli potrà sempre tornare alla situazione precedente l'ultima mossa attraverso il comando UNDO). Se la porta è chiusa a chiave, invece, la trasformazione riesce (in questo caso togliamo l'attributo `wear` dai `vestiti` e lo diamo al `costume`). Abbiamo aggiunto anche un rifiuto speciale a cambiarsi d'abito al buio, forzando il giocatore ad accendere la luce e quindi, speriamo, a trovare la moneta. Ed infine abbiamo aggiunto il caso di `default`; ricorderete che, in un'istruzione `switch`, questo caso viene chiamato per ogni valore non specificato dell'espressione sotto controllo – in questo caso per la variabile `location`. Dal momento che abbiamo già specificato tutte le possibili locazioni del gioco, questo caso appare solo come misura difensiva, nel caso in cui accada qualcosa d'inaspettato (ad esempio, potremmo decidere di estendere il gioco con una nuova stanza e dimenticare di aggiungerla in questa istruzione `switch`). In condizioni normali e sotto controllo, non dovrebbe essere mai raggiunto, ed in ogni caso non fa male a nessuno avere un pezzetto di codice in più di verifica.

L'azione `wear` si limita a controllare se i vestiti sono già stati indossati, per offrire due tipi di risposte: l'obiettivo del gioco è quello di cambiarsi con il nostro costume da eroe, dopodiché impedire al giocatore di ritornare a vestire i suoi panni abituali. In questo modo alla fine abbiamo per le mani il nostro Capitan Fato in tutta la stupefacenza del suo costume:

```
[TYPE]
    Object  costume "il tuo costume"
        with
            name 'capitan' 'fato' 'costume' 'tuta',
            description
                "Manifattura allo STATO DELL'ARTE, 100% COTONELASTICO(tm)
                 rinforzato chimicamente.",
            before [;
                Wear:
                    if(self has worn) "Sei già vestito da Capitan Fato.";
                    else
                        "Prima dovresti toglierti gli ordinari e ANONIMI
                         vestiti di John Covarth.";
                Disrobe,Change:
                       if (self has worn)
                              "Hai BISOGNO del tuo costume per combattere
                               il crimine!";
                       else   "Non lo stai indossando!";
                Drop, Insert, PutOn:
                    "Il tuo UNICO costume multicolore da Capitan FATO? Il
                     più desiderato capo d'abbigliamento in tutta la
                     città? Certamente NO!";
            ],
        has clothing proper;
```

Notate che abbiamo intercettato l'azione INDOSSA COSTUME e suggerito al giocatore di provare invece TOGLI I VESTITI. Inoltre non permetteremo al giocatore di togliersi il costume una volta che lo abbia indossato, e certamente gli impediremo di lasciarlo da qualche parte, rifiutando il comando `Drop` (posa), `Insert` (inserisci, metti nel) e `PutOn` (metti su).

Incartiamo il regalo

Ci siamo quasi; ancora un paio di piccoli ritocchi ed avremo finito.

La routine Initialise

Ora che tutti gli oggetti del nostro gioco sono stati definiti, dobbiamo aggiungere un paio di linee di codice alla routine `Initialise` per essere sicuri che il giocatore non cominci il gioco nudo:

```
[TYPE]
    [ Initialise;
        #Ifdef DEBUG; pname_verify(); #Endif;     ! suggerito da pname.h
        location = strada;
        move costume to player;
        move vestiti to player; give vestiti worn;
        lookmode = 2;
        "^^Impersonando il tranquillo John Covarth, assistente garzone
         in una insignificante drogheria, ti FERMI di colpo quando il
         tuo udito finissimo decifra una chiamata radio della POLIZIA.
         Un FOLLE sta attaccando la popolazione al Parco Granaio! Devi
         indossare velocemente il tuo costume da Capitan FATO...!^^";
    ];
```

Ricordate che abbiamo incluso la libreria aggiuntiva, `pname`, per eliminare la confusione tra i nomi? Ci sono alcuni commenti aggiuntivi nel file di testo che la accompagna che devono essere presi in considerazione:

> `pname.h` prevede una routine `pname_verify`. Quando la modalità DEBUG è attiva, si può richiamare `pname_verify()` nella routine `Initialise()` per verificare le proprietà `pname` del vostro oggetto.

L'autore della libreria aggiuntiva ha creato uno strumento di debug (una routine) per controllare i possibi errori nell'uso della libreria stessa, e ci raccomanda di usarlo. Quindi includiamo le linee di codice suggerite nella nostra routine `Initialise`:

```
    #Ifdef DEBUG; pname_verify(); #Endif;    ! suggerito da pname.h
```

Come spiega il testo, ciò che fa questo codice è: innanzitutto controllare se il gioco è stato compilato con la modalità Debug attiva (i giochi vengono compilati di default in modalità Strict (severa), che include il Debug); in tal caso, eseguire la routine `pname_verify`, verificando così che tutte le proprietà `pname` siano scritte correttamente.

La morte del nostro eroe

Abbiamo creato tre possibili conclusioni in questo gioco:

1. Il giocatore cerca di cambiarsi nel gabinetto senza che la porta sia chiusa a chiave.
2. Il giocatore prova a colpire Benny mentre sta indossando il costume.
3. Il giocatore riesce ad uscire dal bar vestito da Capitan Fato.

Le conclusioni numero (1) e (2) corrispondono ad una partita persa, mentre la (3) è la soluzione vincente del gioco. La libreria di default prevede per le partite perse o vinte la visualizzazione di due messaggi, rispettivamente,

```
*** Sei morto ***
*** Hai vinto ***
```

Questi messaggi corrispondono ai valori della variabile `deadflag`: 1 per la sconfitta, 2 per la vittoria. Naturalmente è possibile cambiare tali messaggi, come nel nostro caso, visto che il nostro eroe in realtà non muore veramente, ma incappa in un DESTINO ben peggiore della morte stessa. Intendiamo, infatti, dargli un messaggio un poco più dettagliato sulla conclusione della partita. Dobbiamo, perciò, definire una routine `DeathMessage` come abbiamo fatto nel gioco di "Guglielmo Tell", per scrivere i nostri messaggi personalizzati ed assegnar loro un valore di `deadflag` maggiore di 2.

```
[TYPE]
    [ DeathMessage;
        if (deadflag == 3)
            print "La tua identità segreta è stata rivelata.";
        if (deadflag == 4)
            print "Sei stato VERGOGNOSAMENTE sconfitto.";
        if (deadflag == 5)
            print "Voli via, diretto a RISOLVERE la SITUAZIONE!";
    ];
```

Grammatica

Infine, abbiamo bisogno di estendere la grammatica esistente, per permettere un paio di cosette. Abbiamo già visto che abbiamo bisogno del verbo CAMBIA (change). Lo creeremo semplicemente in questo modo:

```
[TYPE]
    [ ChangeSub;
        if (noun has pluralname) print "Non sono";
        else                     print "Non è";
        " qualcosa che devi cambiare per risolvere la situazione.";
    ];

    Verb 'cambia'
        * noun                   -> Change;
```

Notate solamente come il verbo gestisce il controllo se il sostantivo dato è plurale o singolare, per visualizzare il giusto pronome.

Un ulteriore dettaglio: quando il giocatore entra nel bar, potrebbe chiedere a Benny un caffè (come noi abbiamo voluto e pesantemente suggerito), un panino o un dolcetto (entrambi menzionati nella descrizione del bar), del cibo o uno stuzzichino (menzionati qui e là, e abbiamo già provveduto a questo); ma cosa accade se il giocatore prova a mangiare una mela? O a rompere le uova? Ci sono così tanti elementi decorativi che possono essere inseriti in un gioco, e caricare il dizionario con il menu completo di Benny, sarebbe un po' strafare.

Qualcuno protrebbe ragionevolmente immaginare che la linea di `default` alla fine dell'azione `Give` nella proprietà `orders` gestisca ogni input che non sia altrimenti specificato:

```
    orders [;
        Give: switch (noun) {
            chiave_del_gabinetto:    codice per la chiave...
            caffe:                   codice per il caffè...
            cibo:                    codice per il cibo...
            menu:                    codice per il menu...
            default:
                "~Non credo sia sul menù, signore.~";
            }
    ],
```

Non è così. La grammatica della libreria che tratta il comando CHIEDI A BENNY ... è questa (specificamente, gli ultimi due punti):

```
Verb 'chiedi' 'domanda'
    *   'a'/'ad'/'all^'/'allo'/'alla'/'al'/'agli'/'ai'/'alle'
        creature
        'circa'/'su'/'sul'/'sui'/'sullo'/'sull^'/'sulla'/'sugli'/
        'sulle'/'di'/'dello'/'della'/'dell^'/'dei'/
        'degli'/'delle' topic
                                            -> Ask
    *   'a'/'ad'/'all^'/'allo'/'alla'/'al'/'agli'/'ai'/
        'alle' creature                     -> Ask
    *   'scusa'/'scuse'                     -> Sorry
    *   'a'/'ad'/'all^'/'allo'/'alla'/'al'/'agli'/'ai'/
        'alle' creature noun                -> Askfor
    *   noun 'a'/'ad'/'all^'/'allo'/'alla'/'al'/'agli'/'ai'/
        'alle' creature
                                            -> AskFor reverse;
```

Come vedete viene preso il `noun`, il che significa che il giocatore deve chiedere a Benny un oggettto del gioco reale presente in quel momento. Assumendo che il giocatore menzioni tale oggetto, l'interprete lo trova nel dizionario e mette il suo ID (identificativo) interno nella variabile `noun`, dove la nostra istruzione `switch` può gestirlo. In questo modo il comando CHIEDI LA CHIAVE A BENNY o CHIEDI A BENNY LA CHIAVE assegna l'oggetto `chiave_del_bagno` alla variabile `noun`, e Benny risponde. Anche CHIEDI A BENNY I CLIENTI funziona, è infatti compreso dal caso di `default`. Ma CHIEDI A BENNY GLI SPAGHETTI ALLA BOLOGNESE non funzionerà: non abbiamo alcun oggetto chiamato Spaghetti alla Bolognese (o quasiasi altra prelibatezza della cucina di Benny) – le parole `'spaghetti'` e `'bolognese'` semplicemente non sono nel dizionario. Questa è forse una delle maggiori deficienze del nostro gioco, ma ci porterà via davvero poco tempo permettere a Benny di usare la sua linea di codice di default per QUALSIASI parola usata dal giocatore.

Abbiamo bisogno di estendere la grammatica esistente del verbo CHIEDI:

```
[TYPE]
    Extend 'chiedi'
        *   'a'/'ad'/'all^'/'allo'/'alla'/'al'/'agli'/'ai'/
            'alle' creature topic               -> Askfor
        *   topic 'a'/'ad'/'all^'/'allo'/'alla'/'al'/'agli'/'ai'/
            'alle' creature
                                                -> AskFor reverse;
```

Questa linea sarà aggiunta alla fine della grammatica esistente di Chiedi, così che non sovrascriverà la normale linea di verifica del sostantivo (`noun`). `topic` è un simbolo che pressappoco significa "quasiasi tipo di input"; il valore di `noun` non è importante, poiché sarà gestito dal caso di `default`. Ora il giocatore può chiedere a Benny un sandwich di tonno o dei divertimenti, ottenendo come risposta: "Non credo che sia sul menù, signore", il che fa di Benny un ottimo barista.

E questo è tutto; sulla risposta leggermente surreale al comando CHIEDI A BENNY DEI DIVERTIMENTI, abbiamo mantenuto lo stile che ci proponevamo scrivendo "Capitan Fato". La guida è quasi completa. Tutto ciò che rimane da dire riguarda la ricapitolazione di alcuni degli argomenti più

importanti e un paio di parole in più sulla compilazione e la correzione dei bug. E quindi potremo gettarvi nel grande e profondo mondo della creazione di IF.

14. Gli ultimi punti brutti

E così abbiamo concluso i nostri tre esempi, mostrandovi tutto ciò di cui avevamo bisogno del linguaggio di Inform per completarli, e abbiamo fatto molte osservazioni su come e perché qualcosa andava fatto. Nonostante ciò, ci sono molte cose che non sono state dette, o che sono state solo accennate. In questo capitolo rivisiteremo gli argomenti chiave e daremo uno sguardo ad alcune delle cose più importanti tra quelle che non sono state trattate, per darvi un'idea dei problemi e delle soluzioni che sono rimaste nel cassetto.

Parleremo anche, in "Leggere il codice di altri autori" più avanti in questo capitolo, dei diversi modi di fare alcune cose che prima abbiamo preferito non accennarvi, ma che è probabile incontrerete guardando il codice scritto da altri programmatori.

Le spiegazioni che daremo vi sembreranno forse un po' monotone, ma credeteci: giocando in questo campo duro e polveroso toccheremo tutto ciò che è fondamentale sapere per cominciare l'apprendimento di Inform. E, come sempre, l'*Inform Designer's Manual* è lì a coprire tutto ciò che non abbiamo trattato.

Espressioni

In questa guida abbiamo usato il termine *espressione* diverse volte; ecco grosso modo che cosa volevamo indicare.

- Una *espressione* è un singolo *valore*, o alcuni *valori* combinati usando gli *operatori* e qualche volta le parentesi `(...)`.
- I possibili *valori* includono:
  - un numero intero (da -32768 a 32767)
  - qualcosa che ha un valore numerico (un carattere `'a'`, una parola del dizionario `'aardvark'`, una stringa `"l'avventura di aardvark"` o una azione `##Look`)
  - l'identificativo interno di una costante, un oggetto, una classe o una routine
  - (solo in un'istruzione durante l'esecuzione del gioco e non in una direttiva durante la compilazione dello stesso) i contenuti di una variabile, o il valore di ritorno di una routine.
- I possibili *operatori* includono:
  - un operatore aritmetico: `+ - * / % ++ --`
  - un operatore logico: `& | ~`

- un operatore numerico di comparazione: `== ~= > < >= <=`
- un operatore condizionale di un oggetto: `ofclass in notin provides has hasnt`
- un operatore booleano: `&& || ~~`

Identificatori interni

Molte delle cose che sono definite nel file sorgente – oggetti, variabili, routine, etc. hanno bisogno di un proprio nome, così che altre istruzioni possano riferirsi ad esse. Chiamiamo questo nome "identificatore interno" (poiché è usato solo all'interno del codice sorgente e non è visibile al giocatore), e usiamo le diciture `obj_id` (o anche `id_oggetto`), `var_id` (o anche `id_variabile`), `routine_id`, etc. per indicare dove deve essere usato. Un ID interno:

- può essere lungo fino a 32 caratteri
- deve iniziare con una lettera o un underscore, e quindi continuare con lettere dalla `A` alla `Z`, underscore `_` e numeri da `0` a `9` (facendo attenzione al fatto che le lettere maiuscole e quelle minuscole vengono trattate allo stesso modo, ovvero non vengono distinte)
- generalmente dovrebbe essere unico in tutti i file, compresi il sorgente, le librerie standard e ogni altra libreria supplementare usata (con l'unica eccezione delle variabili locali delle routine che non sono visibili all'esterno della stessa).

Istruzioni

Una **istruzione** è un comando all'interprete, che gli dice cosa fare durante l'esecuzione del gioco. *Deve* essere data usando lettere minuscole, e deve sempre terminare con un punto e virgola. Alcune istruzioni, come `if`, controllano una o più altre istruzioni. Usiamo la dicitura *`blocco_istruzioni`* per rappresentare sia una singola *`istruzione`*, sia un qualsiasi numero di *`istruzioni`* racchiuse tra parentesi graffe:

```
istruzione;
{ istruzione; istruzione; .... istruzione; }
```

Le istruzioni che abbiamo incontrato

Nei nostri esempi abbiamo usato le seguenti istruzioni, quasi la metà di quelle offerte da Inform:

```
give obj_id attributo;
give obj_id attributo attributo ... attributo;
 if (espressione) blocco_istruzioni
```

```
if (espressione) blocco_istruzioni else blocco_istruzioni
move obj_id to genitore_obj_id;
objectloop (var_id) blocco_istruzioni
print valore;
print valore, valore, ... valore;
print_ret valore;
print_ret valore, valore, ... valore;
remove obj_id;
return false;
return true;
style underline; print...; style roman;
switch (espressione) {
    valore: istruzione; istruzione; ... istruzione;
    valore: istruzione; istruzione; ... istruzione;
    ...
    default: istruzione; istruzione; ... istruzione;
}
"stringa";
"stringa", valore, ... valore;
<azione>;
<azione noun>;
<azione noun second>;
<<azione>>;
<<azione noun>>;
<<azione noun second>>;
```

Istruzioni che non abbiamo incontrato

Sebbene nei nostri esempi non ne abbiamo avuto bisogno, queste istruzioni cicliche sono spesso utili:

break;

continue;

do *blocco_istruzioni* **until** (*espressione*)

for (*imposta_variabile:ripeti_mentre_espressione:aggiorna_variabile*)
blocco_istruzioni

while (*espressione*) *blocco_istruzioni*

D'altra parte, vi suggeriamo di lasciare da parte per il momento questo tipo di istruzioni:

```
box
font
jump
newline
spaces
string
```

In particolare, evitate di usare l'istruzione deprecata `jump` se vi è possibile.

Le regole dell'istruzione di stampa Print

Nelle istruzioni `print` e `print_ret` , ogni *`valore`* può essere:

- una *`espressione`* numerica, visualizzata come numero decimale con segno,
- una *`"stringa"`*, visualizzata alla lettera o
- una regola di stampa. Potete crearne una vostra, o usarne una standard, incluse:

`(a)` *`obj_id`* : il nome dell'oggetto, preceduto da "un" "una" "uno" "un'" o "alcune" "alcuni" a seconda di genere e numero.

`(the)` *`obj_id`* : il nome dell'oggetto preceduto da "il" "lo" "la" "i" "gli" "le" "l'" a seconda di genere e numero.

`(The)` *`obj_id`* : il nome dell'oggetto preceduto da "Il" "Lo" "La" "I" "Gli" "Le" a seconda di genere e numero.

`(number)` *`expression`* – il valore numerico di una espressione in parole

`(thatorthose)` *`obj_id`*: stampa "quello" "quella" "quelli" "quelle" a seconda di genere e numero dell'oggetto.

`(cthetorthose)` *`obj_id`*: stampa "Quello" "Quella" "Quelli" "Quelle" a seconda di genere e numero dell'oggetto.

`(itorthem)` *`obj_id`*: stampa "lo" "la" "li" "le" a seconda di genere e numero dell'oggetto.

`(isorare)` *`obj_id`*: stampa "sono" "è" a seconda del numero dell'oggetto.

`(cisorare)` *`obj_id`*: stampa "Sono" "È" a seconda del numero dell'oggetto.

`(WhomorWhich1)` *`obj_id`*: stampa "lle quali" "i quali" "lla quale" "l quale" a seconda di genere e numero dell'oggetto.

`(whomorwhich2) obj_id`: stampa "il quale" "la quale" "i quali" "le quali" a seconda di genere e numero dell'oggetto.

`(artda) obj_id`: stampa la preposizione articolata "da" più l'articolo, ad esempio "dallo", "dalla", etc.

`(artsu) obj_id`: stampa la preposizione articolata "su" più l'articolo, ad esempio "sullo", "sulla", etc.

`(artin) obj_id`: stampa la preposizione articolata "in" più l'articolo, ad esempio "nello", "nella", etc.

`(arta) obj_id`: stampa la preposizione articolata "a" più l'articolo, ad esempio "allo", "alla", etc.

`(artdi) obj_id`: stampa la preposizione articolata "di" più l'articolo, ad esempio "dello", "della", etc.

Direttive

Una **direttiva** è intesa come un'istruzione per il compilatore, che gli dice cosa fare al momento della compilazione, mentre il file sorgente viene tradotto in Z-code. Per convenzione viene data con l'iniziale maiuscola (sebbene il compilatore non lo richieda obbligatoriamente) e finisce sempre con un punto e virgola.

Le direttive che abbiamo incontrato

Abbiamo usato tutte queste direttive; notate che per `Class`, `Extend`, `Object` e `Verb` la sintassi completa supportata è più sofisticata di quella in forma base che abbiamo presentato in questa guida:

```
Class class_id
 with   proprietà valore,
        proprietà valore,
        ...
        proprietà valore,
 has attributo attributo ... attributo;
 Constant const_id;
 Constant const_id = espressione;
 Constant const_id espressione;
 Extend 'verbo'
     * token token ... token -> azione
     * token token ... token -> azione
     ...
     * token token ... token -> azione;
```

```
Include "nome_del_file";

Object obj_id "nome_esterno" genitore_obj_id
    with   proprietà valore,
           proprietà valore,
           ...
           proprietà valore,
    has attributo attributo ... attributo;
class_id obj_id " nome_esterno " genitore_obj_id
    with   proprietà valore,
           proprietà valore,
           ...
           proprietà valore,
    has attributo attributo ... attributo;
Release espressione;
Replace routine_id;
Serial "aammgg";
Verb 'verbo'
    * token token ... token -> azione
    * token token ... token -> azione
    ...
    * token token ... token -> azione;
! testo di commento che viene ignorato dal compilatore
[ routine_id; istruzione; istruzione; ... istruzione; ];
#Ifdef qualsiasi_id; ... #Endif;
```

Direttive che non abbiamo incontrato

Vi sono solo una manciata di direttive davvero utili che non abbiamo avuto necessità di usare:

```
Attribute attributo;
Global var_id;
Global var_id = espressione;
Property proprietà;
Statusline score;
Statusline time;
```

Ma ve ne sono molte che non abbiamo visto e che a questo livello possono essere per ora trascurate:

```
Abbreviate
Array
```

```
Default
End
Ifndef
Ifnot
Iftrue
Iffalse
Import
Link
Lowstring
Message
Switches
System_file
Zcharacter
```

Oggetti

Un oggetto è in realtà una semplice collezione di variabili che assieme rappresentano le capacità e lo stato attuale di alcune componenti specifiche del modello del mondo. Tutte le variabili vengono chiamate proprietà; le variabili più semplici – che possono adottare solo due valori – vengono dette attributi.

Le proprietà

La libreria definisce circa quarantotto variabili proprietà standard (come ad esempio `before` o `name`), ma se ne possono creare anche altre semplicemente usandole nella definizione di un oggetto.

In particolare potete creare ed inizializzare una proprietà nel segmento dell'oggetto introdotto dalla parola chiave `with`:

```
proprietà, ! impostata al valore zero/falso
proprietà valore, ! impostata ad un singolo valore
proprietà valore valore ... valore,    ! impostata con più valori
```

In ogni caso, il *valore* è o una *espressione* nel momento della compilazione, o una routine incorporata:

```
proprietà espressione,
proprietà [; istruzione; istruzione; ... istruzione; ],
```

Potete riferirvi al valore di una proprietà in questo modo:

```
self.proprietà ! solo all'interno dello stesso oggetto
obj_id.proprietà ! ovunque
```

e potete testare se la definizione di un oggetto include una certa proprietà così:

```
(obj_id provides proprietà) ! è vera o falsa
```

Attributi

La libreria definisce circa trenta attributi standard (come ad esempio `open` o `worn`); la creazione di attributi supplementari è richiesta piuttosto raramente.

Potete inizializzare degli attributi nel segmento di un oggetto introdotto dalla parola chiave `has`:

```
attributo attributo ...      ! impostazione iniziale
~attributo ~attributo ...    ! inizialmente non impostato (valore di
                             ! default)
```

Potete impostare o pulire gli attributi in questo modo:

```
give obj_id attributo attributo ... attributo;
give obj_id ~attributo ~attributo ... ~attributo;
```

e potete testare il valore corrente di un attributo così:

```
(obj_id has attributo)     ! è vero o falso
(obj_id hasnt attribute)   ! è falso o vero
```

Classi

Potete testare se un oggetto è un membro di una determinate classe:

```
(obj_id ofclass class_id) ! è vero o falso
```

L'albero degli oggetti

Potete specificare il genitore (`parent`) di un oggetto (ad es. la sua locazione all'inizio del gioco) come parte della definizione dell'oggetto:

```
Object obj_id "nome_esterno " genitore_obj_id
    with ...
```

Esiste anche un'altra sintassi che utilizza frecce -> -> come queste, che vengono anche usate per stabilire il genitore iniziale di un oggetto. Le spiegheremo in "Leggere il codice di altri autori", più avanti.

Potete modificare la posizione di un oggetto all'interno dell'albero:

```
move obj_id to genitore_obj_id;
```

e potete muovere un oggetto al di fuori dell'albero così che non abbia alcun genitore (`parent`):

```
remove obj_id;
```

Dato l'*obj_id* dell'oggetto, potete determinare quale sia il genitore (`parent`) dell'oggetto, il figlio più vecchio (`child`), e il fratello (`sibling`, ovvero oggetto che ha lo stesso genitore) immediatamente più giovane – l'oggetto adiacente. Potete anche contare quanti figli (`children`) immediati possiede:

```
parent(obj_id)
```

```
child(obj_id)
sibling(obj_id)
children(obj_id)
```

Potete testare se un *obj_id* è un figlio (child) di un altro oggetto:

```
(obj_id in genitore_obj_id)     ! è vero o falso
(obj_id notin genitore_obj_id)  ! è falso o vero
```

Se avete bisogno di sapere se un oggetto è un figlio (child), o un nipote (*grandchild*), o un pronipote (*greatgrandchild*), etc. di un altro oggetto, usate il comando:

```
IndirectlyContains(genitore_obj_id, obj_id)   ! è vero o falso
```

Infine potete determinare l'oggetto (se esiste) di cui due oggetti sono allo stesso momento figli (child) o nipoti (*grandchildren*), o pronipoti (*great-grandchildren*), etc. usando il comando:

```
CommonAncestor(obj_id1, obj_id2)
```

Routine

Inform prevede routine indipendenti e routine incorporate.

Routine indipendenti

Le routine indipendenti sono definite in questo modo:

```
[ routine_id; istruzione; istruzione; ... istruzione; ];
```

e vengono richiamate nel codice così:

```
routine_id()
```

Le routine incorporate

Esse sono incorporate come il valore della proprietà di un oggetto :

```
proprietà [;istruzione; istruzione; ... istruzione; ],
```

e sono normalmente richiamate automaticamente dalla libreria, o manualmente in questo modo:

```
self.proprietà()     ! solo all'interno dello stesso oggetto
obj_id.proprietà ()  ! ovunque
```

Argomenti e variabili locali

Entrambi i tipi di routine supportano fino a quindici variabili locali – ossia variabili che possono essere usate solo dalle istruzioni all'interno della routine, e che sono automaticamente impostate a zero ogni volta che la routine viene richiamata:

```
[ routine_id var_id ... var_id; istruzione; ... istruzione; ];
proprietà [ var_id ... var_id; istruzione; ... istruzione; ],
```

Potete passare ad una routine indipendente fino a 7 argomenti, elencandoli tra le parentesi quando richiamate la routine; ad una routine incorporata invece possono essere passati fino ad un massimo di 5 argomenti. L'effetto è semplicemente quello d'impostare le variabili locali corrispondenti ai valori degli argomenti invece che a zero:

```
routine_id(espressione, espressione, ... espressione)
```

Sebbene funzioni, questa tecnica è raramente utilizzata nelle routine incorporate, poiché non vi è un meccanismo nella libreria che possa fornire automaticamente i valori degli argomenti quando richiama la routine.

Valori di ritorno

Ogni routine restituisce un singolo valore, che può essere fornito sia esplicitamente in alcune forme dall'istruzione `return`:

```
[ routine_id; istruzione; istruzione; ... return expr; ];
! restituisce expr

proprietà [; istruzione; istruzione; ... return expr; ],
! restituisce expr
```

sia implicitamente quando la routine finisce le sue istruzioni. Se nessuna di queste istruzioni è un – `return`, `print_ret`, `"..."` o `<<...>>` – che causa un esplicito ritorno (`return`), allora:

```
[ routine_id; istruzione; istruzione; ... istruzione; ];
```

restituisce (`return`) il valore `true` e

```
proprietà [; istruzione; istruzione; ... istruzione; ]
```

restituisce (`return`) il valore `false`.

Questa differenza è *importante*. Ricordatevi: lasciate a se stesse, le routine indipendenti restituiscono (`return`) il valore `true`, le routine incorporate restituiscono (`return`) il valore `False`.

Ecco un esempio di routine indipendente che restituisce il valore maggiore dei due argomenti che gli sono stati passati:

```
[ Max a b; if (a > b) return a; else return b; ];
```

e qui abbiamo alcuni esempi del suo uso (notate che il primo esempio sebbene sia corretto, non fa niente di utile):

```
Max(x,y);
x = Max(2,3);
if (Max(x,7) == 7) ...
switch (Max(3,y)) { ...
```

Le routine della libreria e le routine punti di entrata (entry points)

Una routine della libreria è una routine indipendente, inclusa nei file della libreria stessa, che potete richiamare dal vostro file sorgente se avete bisogno delle funzionalità che mette a disposizione tale routine.

Abbiamo menzionato le seguenti routine:

```
IndirectlyContains(parent_obj_id, obj_id)
PlaceInScope(obj_id)
PlayerTo(obj_id, flag)
StartDaemon(obj_id)
StopDaemon(obj_id)
```

Diversamente, una routine che funge da punto di entrata è una routine che si può inserire nel file sorgente, e che in questo caso viene chiamata dalla libreria in uno specifico momento. Abbiamo menzionato nei nostri esempi le seguenti routine punti di entrata:

```
DeathMessage()
InScope(attore_obj_id)
```

E questa come unica routine obbligatoria:

```
Initialise()
```

Vi è una lista completa in "Le Routine della Libreria" e in "Punti di entrata opzionali" nell'appendice F.

Leggere il codice di altre persone

Giusto all'inizio di questa guida vi abbiamo avvertito che questo lavoro per forza di cose non può essere esauriente; ci siamo concentrati sulla presentazione degli aspetti più importanti di Inform, cercando di essere il più chiari possibile. Naturalmente, leggendo l'*Inform Designer's Manual*, e più specificatamente andando a vedere i sorgenti di giochi completi o delle librerie opzionali prodotti da altri autori, vi imbatterete in altri modi di scrivere il codice in Inform – ed è possibile che voi, come altri autori, preferiate un altro approccio, diverso dai

nostri metodi. La cosa importante è trovare uno stile che sia adatto a se stessi e che sia efficace a raggiungere gli obiettivi che vi proponete. In questa sezione metteremo in risalto alcune delle differenze più evidenti che potreste trovare in altri lavori rispetto al nostro approccio.

Il layout del codice

Ogni autore ha il suo stile nello scrivere il codice sorgente, e solitamente sono tutti peggiori di quello che avete adottato voi. La flessibilità di Inform rende semplice per un programmatore scegliere lo stile che lo soddisfa maggiormente; sfortunatamente, per alcuni autori questa scelta sembra essere influenzata dalla scuola d'arte di Jackson Pollock. Noi vi consigliamo di essere coerenti, di usare sempre al massimo gli spazi bianchi e l'indentazione, di scegliere nomi chiari per oggetti, variabili e routine, di aggiungere commenti alle sezioni difficili da comprendere, di *pensare* attivamente, mentre scrivete il codice, su come renderlo il più comprensibile possibile.

Tale esigenza risulterà ancora più chiara andando a dare un sguardo al codice di alcune librerie opzionali prodotte dalla comunità. Questo esempio, con i nomi alterati, proviene dritto dritto dall'Archivio:

```
[xxxx i j;
if (j==0) rtrue;
if (i in player) rtrue;
if (i has static || (i has scenery)) rtrue;
action=##linktake;
if (runroutines(j,before) ~= 0 || (j has static || (j has
scenery))) {
print "You'll have to disconnect ",(the) i," from ",(the) j,"
first.^";
rtrue;
}
else {
if (runroutines(i,before)~=0 || (i has static || (i has scenery)))
{
print "You'll have to disconnect ",(the) i," from ",(the) j,"
first.^";
rtrue;
}
else
if (j hasnt concealed && j hasnt static) move j to player;
if (i hasnt static && i hasnt concealed) move i to player;
action=##linktake;
if (runroutines(j,after) ~= 0) rtrue;
print "You take ",(the) i," and ",(the) j," connected to it.^";
rtrue;
}
];
```

Ecco la stessa routine dopo alcuni minuti spesi unicamente a renderla più comprensibile; non abbiamo provveduto a testare se (ancora) funziona, anche se il secondo `else` sembra sospetto:

```
[ xxxx i j;
       if (i in player || i has static or scenery || j == nothing)
              return true;
       action = ##LinkTake;
       if (RunRoutines(j,before) || j has static or scenery)
              "You'll have to disconnect ", (the) i, " from ",
              (the) j, " first.";
       else {
           if (RunRoutines(i,before) || i has static or scenery)
              "You'll have to disconnect ", (the) i, " from ",
              (the) j, " first.";
           else
              if (j hasnt static or concealed) move j to player;
           if (i hasnt static or concealed) move i to player;
           if (RunRoutines(j,after)) return true;
           "You take ", (the) i, " and ", (the) j, " connected to
           it.";
       }
];
```

Speriamo che siate d'accordo con noi che il risultato vale la pena di un piccolo lavoro extra. Il codice viene scritto una volta, ma viene letto dozzine e dozzine di volte.

Scorciatoie

Ci sono alcune istruzioni brevi, alcune più utili di altre, nelle quale vi potreste imbattere.

- Queste cinque linee fanno la stessa cosa:

```
return true;
return 1;
return;
rtrue;
]; ! alla fine di una routine indipendente
```

- Queste quattro linee fanno la stessa cosa:

```
return false;
return 0;
rfalse;
]; ! alla fine di una routine incorporata
```

- Queste quattro linee fanno la stessa cosa:

```
print "stringa"; new_line; return true;
print "stringa^"; return true;
print_ret "stringa";
"stringa";
```

- Queste linee sono identiche:

```
print valore1; print valore2; print valore3;
print valore1, valore2, valore3;
```

- Queste linee sono identiche:

```
<azione noun second>; return true;
<<azione noun second>>;
```

- Anche queste linee sono identiche:

```
print "^";
new_line;
```

- Queste istruzioni `if` sono equivalenti:

```
if (MiaVar == 1 || MiaVar == 3 || MiaVar == 7) ...
if (MiaVar == 1 or 3 or 7) ...
```

- Queste istruzioni if sono equivalenti:

```
if (MiaVar ~= 1 && MiaVar ~= 3 && MiaVar ~= 7) ...
if (MiaVar ~= 1 or 3 or 7) ...
```

- In una istruzione `if` , l'argomento tra parentesi può essere qualsiasi tipo di espressione; tutto ciò che conta è il suo valore: zero (falso) o qualsiasi altra cosa (vero). Per esempio queste istruzioni sono equivalenti:

```
if (MiaVar ~= false) ...
if (~~(MiaVar == false)) ...
f (MiaVar ~= 0) ...
if (~~(MiaVar == 0)) ...
if (MiaVar) ...
```

  Notate che la seguente istruzione testa specificatamente se *MiaVar* contiene il valore vero (1), e *non* se il suo valore è qualcosa di diverso da zero.

```
if (MiaVar == true) ...
```

- Se *MiaVar* è una variabile, le istruzione `MiaVar++;` e `++MiaVar;` funzionano esattamente come `MyVar = MiaVar + 1;` Per esempio, queste linee sono equivalenti:

```
MiaVar = MiaVar + 1; if (MiaVar == 3) ...
if (++MiaVar == 3) ...
if (MiaVar++ == 2) ...
```

  Ciò che accomuna `MiaVar++` e `++MiaVar` è che entrambe aggiungono uno a `MiaVar`. Ciò che le differenzia è il valore che viene ritornato (`return`): `MiaVar++` restituisce il valore corrente di `MiaVar` e quindi effettua l'incremento, mentre `++MiaVar` esegue il "+1" prima e quindi riporta (`return`) il valore incrementato.

  Nell'esempio, se `MiaVar` correntemente contiene 2 allora `++MiaVar` restituisce (`return`) 3 e `MiaVar++` restituisce 2, sebbene in entrambi i casi il

valore di `MiaVar` alla fine sia 3. Come altro esempio, prendiamo questo codice (da Helga in "Guglielmo Tell"):

```
Talk:
    self.frasi_dette = self.frasi_dette + 1;
    switch (self.frasi_dette) {
        1:      score = score + 1;
                print_ret "Ringrazi calorosamente Helga per la
                mela.";
        2:      score = score + 1;
                print_ret "~Ci vediamo presto.~";
        default: return false;
    }
],
```

esso potrebbe essere riscritto più succintamente in questo modo:

```
Talk:
    switch (++self.frasi_dette) {
              1:  score++;
                  print_ret "Ringrazi calorosamente Helga per la
                  mela.";
              2:  score++;
                  print_ret "~Ci vediamo presto.~";
              default: return false;
          }
],
```

- Allo stesso modo, le istruzioni `MiaVar--;` e `--MiaVar;` funzionano come `MiaVar = MiaVar - 1;` Ancora una volta queste linee sono equivalenti:

```
MiaVar = MiaVar - 1; if (MiaVar == 7) ...

if (--MiaVar == 7) ...

if (MiaVar-- == 8) ...
```

Proprietà "number" e attributo "General"

La libreria definisce una proprietà standard `number` e un attributo standard `general` per ogni oggetto, i cui ruoli sono indefiniti: sono variabili dallo scopo generico a disposizione del programmatore per i suoi più reconditi desideri.

Vi raccomandiamo di evitare l'uso di questi due tipi di variabili, soprattutto perché i loro nomi sono, per loro precisa natura, così insulsi da essere per lo più senza senso. Il vostro gioco risulterà molto più chiaro e semplice in fase di debug se vi avvarrete di nuove variabili proprietà - con i nomi appropriati - come parte delle definizioni dei vostri Oggetti e Classi.

Proprietà e attributi comuni

Come alternative alla creazione di nuove proprietà individuali, che si applicano solo ad un singolo oggetto (o classe di oggetti), è possibile inventare nuove proprietà e attributi che, come quelli definiti dalla libreria, sono disponibili per tutti gli oggetti. La necessità di eseguire simili operazioni è piuttosto rara, ed è

per lo più usata nelle librerie supplementari (come ad esempio l'estensione alla libreria `pname.h` che abbiamo incontrato in "Capitan FATO: terza!" nel Capitolo 12, che da ad ogni oggetto la proprietà `pname` e un attributo `phrase_matched`).

Per crearli, dovreste usare queste direttive vicino l'inizio del codice sorgente:

```
Attribute  attributo;
Property   proprietà;
```

Vi raccomandiamo d'evitare di sfruttare queste due direttive senza una reale necessità del loro uso all'interno del gioco. Esiste un limite di quarantotto attributi (dei quali correntemente la libreria ne definisce trenta) e sessantadue proprietà comuni (delle quali la libreria correntemente ne definisce quarantotto).

D'altra parte, il numero di proprietà individuali che potete aggiungere è virtualmente illimitato.

Costruire l'albero degli oggetti

Per tutta questa guida, abbiamo definito la posizione iniziale di ogni oggetto nell'insieme di tutti gli oggetti (albero degli oggetti) menzionando esplicitamente il genitore (`parent`), nel caso vi fosse, nella prima linea della definizione di ogni oggetto – ciò che abbiamo chiamato intestazione – o, per pochi oggetti che spuntavano fuori in più di un posto usando la loro proprietà `found_in`. Per esempio in "Guglielmo Tell" abbiamo definito ventisette oggetti; omettendo quelli che usano la proprietà `found_in` per definire la loro posizione all'inizio del gioco, abbiamo definito la posizione iniziale degli oggetti in questo modo:

```
Room strada "Una strada di Altdorf"
Room vicino_piazza "Lungo la strada"
Furniture banco "banco di frutta e verdura" vicino_piazza
Prop "patate" vicino_piazza
Prop "frutta e verdura" vicino_piazza
NPC proprietaria "Helga" vicino_piazza
Room piazza_sud "Lato sud della piazza"
Room centro_piazza "In mezzo alla piazza"
Furniture palo "palo di legno" centro_piazza
Room piazza_nord "Lato nord della piazza"
Room mercato "Piazza del mercato"
Object albero "tiglio" mercato
    NPC balivo "balivo" mercato
    Object arco "arco"
    Object faretra "faretra"
    Arrow "freccia" faretra
    Arrow "freccia" faretra
```

```
Arrow "freccia" faretra

Object mela "mela"
```

Alcuni di questi oggetti cominciano il gioco come genitori (`parent`): `vicino_piazza`, `centro_piazza`, `mercato` e `faretra` tutti hanno oggetti figli (`child`) sotto di essi: questi figli (`children`) menzionano il loro genitore (`parent`) come ultima voce nella prima riga di definizione dell'oggetto (*intestazione*).

Esiste una sintassi alternativa capace di ottenere lo stesso albero di oggetti, usando delle "frecce". Ecco come potremmo alternativamente definire le relazioni di parentela (`child-parent`) tra gli oggetti:

```
Room strada "Una strada di Altdorf"

Room vicino_piazza "Lungo la strada"
Furniture -> banco "banco di frutta e verdura"
Prop -> "patate"
Prop -> "frutta e verdura"
NPC -> proprietaria "Helga"

Room piazza_sud "Lato sud della piazza"

Room centro_piazza "In mezzo alla piazza"
Furniture -> palo "palo di legno"
Room piazza_nord "Lato nord della piazza"

Room mercato "Piazza del mercato"
Object -> albero "tiglio"
NPC -> balivo "balivo"

Object arco "arco"

Object faretra "faretra"
Arrow -> "freccia"
Arrow -> "freccia"
Arrow -> "freccia"

Object mela "mela"
```

L'idea è che le informazioni dell'intestazione di un oggetto o iniziano con una freccia, o finiscono con un *`obj_id`*, o nessuna delle due (entrambe le forme allo stesso tempo non sono permesse). Un oggetto con nessuna delle due diciture non ha genitore: in questo esempio, abbiamo tutte le locazioni e anche l'arco, la faretra (che viene spostata nell'oggetto player nella routine `Initialise`) e la mela (che rimane senza genitore fino a quando Helga non la da a Guglielmo).

Un oggetto che comincia con una freccia singola `->` è definito come figlio (`child`) dell'oggetto che lo precede più vicino senza genitore (`parent`). Così nell'esempio, l'oggetto albero e balivo sono entrambi figli (`children`) del mercato.

Per definire un figlio (`child`) di un figlio (`child`), devono essere utilizzate due frecce `-> ->`, e così via. In "Guglielmo Tell", questa situazione non s'incontra; per illustrare come funziona, immaginiamo che all'inizio del gioco le patate e gli altri ortaggi siano *sul* banco. Avremmo quindi dovuto usare:

```
Room vicino_piazza "Lungo la strada"
Furniture -> banco "banco di frutta e verdura"
Prop -> -> "patate"
```

```
Prop -> -> "frutta e verdura"
NPC -> proprietaria "Helga"
...
```

In questo modo, gli oggetti con una freccia (il `banco` e la `proprietaria`) sono figli (`children`) dell'oggetto precedente più vicino senza genitore (`parent`) (ovvero la locazione), e gli oggetti con due frecce (i vegetali) sono figli dell'oggetto precedente più vicino definito con una singola freccia (il `banco`).

I vantaggi di usare le frecce includono che:

- Si è costretti a definire gli oggetti in un ordine "razionale".
- Si devono usare meno *`obj_id`* (sebbene in questo gioco la cosa non faccia molta differenza).

Gli svantaggi invece sono:

- Il fatto che gli oggetti sono vincolati alla posizione della loro definizione non è necessariamente un sistema più intuitivo per tutti i programmatori.
- Specialmente in una locazione affollata, è più difficile essere certi di come le varie relazioni di parentela siano impostate, oltre il dover contare con attenzione un gran numero di frecce.
- se si sposta un genitore nella gerarchia iniziale ad un livello più alto o più basso, bisogna necessariamente andare a cambiare tutti i suoi figli (`children`) aggiungendo o rimuovendo frecce; ciò non si rende necessario quando il genitore (`parent`) è invece indicato nell'intestazione dell'oggetto figlio (`child`).

Noi preferiamo esplicitare il nome del genitore (`parent`), ma ricordate che incontrerete sicuramente entrambe le forme.

Le virgolette nella proprietà "name"

Abbiamo già spiegato in "Le cose tra virgolette" nel Capitolo 4 la differenza tra virgolette `"..."` (stringhe che devono essere stampate) ed apici `'...'` (parole del dizionario). Sfortunatamente qualche volta Inform potrebbe trarvi in inganno in questa distinzione: potrete infatti trovare in alcuni codici le virgolette nella propretà `name` e nelle direttive `Verb`:

```
NPC proprietaria "Helga" vicino_piazza
       with name  "proprietaria"       "verduraia"       "venditrice"
                  "negoziante"    "mercante"    "Helga"    "vestito"
                  "sciarpa",
       ...

Verb "chiacchera" "conversa" "c//"
   *   "con" creature
                    -> Talk;

Extend "parla" first
   *   "a"/"ad"/"all^"/"allo"/"alla"/"a"/
       "agli"/"ai"/"alle"/"con" creature
                    -> Talk;
```

PER FAVORE non utilizzare le virgolette in questi casi. Non farete che confondere voi stessi: queste sono parole di dizionario, non stringhe; rendete le cose semplici - e più chiare possibile - in modo da non confondervi con la punteggiatura.

Usi obsoleti

Infine, ricordate che Inform è stato sviluppato a partire dal 1993. In tutto questo tempo, Graham si è preoccupato di mantenere il più possibile la compatibilità con le vecchie versioni così che i giochi scritti anni fa, per versioni precedenti del compilatore e delle librerie, possano essere ancora compilati. Generalmente questa è considerata una buona cosa, ma ha i suoi svantaggi portando con sé una quantità di vecchi comandi abbandonati che ancora ci rimangono tra i piedi. Potreste ad esempio vedere giochi che usano la direttiva `Nearby` (che denota la parentela, allo stesso modo di `->`) e la condizione `near` (ossia lo stesso genitore (`parent`)) o ancora con `"\"` che controlla la fine della riga in lunghe istruzioni di stampa `print`. Provate ad interpretarle, cercando però di non usarle.

15. Compilate il vostro gioco

Raro quasi quanto un alchimista che ottiene l'oro dal piombo, il processo di compilazione è quello che trasforma il vostro codice sorgente in uno story file (sebbene il risultato che si ottiene più spesso è un rimprovero che spiega perché ciò non sia – *ancora una volta* – accaduto). Una tale magia è prodotta dal compilatore, che è il programma che prende il vostro codice (più o meno comprensibile) e lo traduce in un file binario: un insieme di numeri che segue uno specifico formato capito solo da un interprete Z-code.

Ad uno sguardo superficiale la compilazione può sembrare un trucco molto semplice. Voi eseguite solo il compilatore, indicandogli qual è il file sorgente dal quale volete generare un gioco et voilà! La magia è fatta.

Tuttavia gli ingredienti dell'incantesimo vanno preparati con cura. Il compilatore "legge" il vostro codice sorgente, ma non nella maniera flessibile con la quale lo leggerebbe un essere umano. È necessario che la sintassi segua alcune regole molto precise, od il compilatore si lamenterà che non è in grado di portare a termine il suo lavoro in queste condizioni. Il compilatore si cura poco del significato, ma molto dell'ortografia, come il più inflessibile degli insegnanti; non ci sono occhi lucidi alla Bambi che tengano qui.

Sebbene l'incantesimo lanciato dal compilatore sia sempre lo stesso, potete indicare, con qualche limitazione, come volete che la magia venga fatta. Ci sono un po' di opzioni che modificano il processo di compilazione; alcune le definite nel codice sorgente, altre con degli *switch* (modificatori) e con determinati comandi quando eseguite il programma. Il compilatore assumerà alcune opzioni predefinite, ma potete sempre modificarle se ne avete la necessità. Molte di queste opzioni sono fornite "nel caso in cui" determinate condizioni speciali debbano essere applicate; altre servono a programmatori con una certa esperienza che hanno bisogno di requisiti complessi ed avanzati, ed è meglio lasciarle (per ora) a quelli che hanno più conoscenze.

Ingredienti

Se il file sorgente non viene scritto in maniera corretta, il compilatore protesta, producendo un messaggio di *warning* (avvertimento) oppure un messaggio di *error* (errore). I warning sono lì per dirvi che potrebbe esserci un errore che potrebbe modificare il comportamento del gioco durante l'esecuzione (run-time), ma il compilatore non fermerà il processo e produrrà, comunque, uno story file. Gli error, invece, riguardano errori che rendono impossibile al compilatore la creazione di tale file. Di questi i *fatal* error (errori fatali) fermano immediatamente la compilazione, mentre i *non-fatal* error (errori non fatali) consentono al compilatore di continuare a leggere il codice sorgente. (Come vedremo tra poco, questo porta sia vantaggi che svantaggi: sebbene possa essere

utile avere un compilatore che riporti quanti più non-fatal error possibili, scoprirete presto che la maggior parte degli stessi potrebbe essere causata da un singolo errore).

Fatal error

È difficile, ma non impossibile, causare un fatal error. Se indicate il nome sbagliato come nome di file sorgente, il compilatore non sarà neanche in grado di partire, dicendovi:

```
Couldn't open source file nomefile
```

[Non posso aprire il file sorgente nomefile]

Se il compilatore individua un gran numero di non-fatal error, può abbandonare l'intero processo con un:

```
Too many errors: giving up
```

[Troppi errori: mi arrendo]

La maggior parte dei fatal error, comunque, capita perché il compilatore termina la memoria o lo spazio su disco; anche se con i computer di oggi è poco comune. Potreste, però, incontrare dei problemi se lo story file, che deve rientrare all'interno dei (ristretti) limiti della Z-Machine, diventa troppo grande. Inform, normalmente, compila il vostro codice sorgente in un file Versione 5 (ecco cosa indica l'estensione .z5 che vedete nel file prodotto), che ha una dimensione massima di 256 Kbyte. Se il vostro gioco è più grande di tale limite, dovete compilare il vostro gioco in un file Versione 8 (.z8), che può crescere fino a 512 Kbyte (e lo potete fare semplicemente impostando lo switch -v8; ma ne parleremo dopo). Avrete bisogno di un'incredibile quantità di codice per superare questi limiti: non dovreste preoccuparvene per qualche mese, o per sempre.

Non-fatal error

I non-fatal error sono molto più comuni. Imparerete subito a conoscere:

```
Expected qualcosa but found qualcos'altro
```

[Mi aspettavo qualcosa ma ho trovato qualcos'altro]

Questo è il modo standard per riportare errori di punteggiatura od errori di sintassi. Se digitate una virgola al posto di un punto e virgola, Inform cercherà invano quello che si aspettava di trovare. La cosa buona è che vi verrà indicata la riga incriminata:

```
Tell.inf(76): Error: Expected directive, '[' or class name but
found found_in
>      found_in
Compiled with 1 error (no output)
```

[Tell.inf(76): Errore: Mi aspettavo una direttiva, '[' od un
nome di classe ma ho trovato found_in
> found_in
Compilato con 1 errore (nessun output)]

Potete vedere il numero di riga (76) e quello che vi è stato trovato, così tornate al codice sorgente e gli date un'occhiata. In questo caso l'errore è stato causato da un punto e virgola dopo la stringa della descrizione (`description`), invece di una virgola:

```
Prop    "banchi"
    with   name 'banchi',
           description   "Cibo, abiti, attrezzature da montagna; la
                          solita roba.";
           found_in strada vicino_piazza,
    has   pluralname;
```

Qui di seguito, riportiamo un messaggio abbastanza ingannevole che potrebbe suggerire che le cose nel nostro codice sorgente sono nell'ordine sbagliato, o che qualche segno di punteggiatura atteso non è stato messo:

```
Fate.inf(459): Error: Expected name for new object or its
textual short name but found door
> Object door
Compiled with 1 error (no output)
```

[Fate.inf(456): Errore: Mi aspettavo il nome per il nuovo
oggetto o la sua descrizione breve ma ho trovato door [porta]
> Object door
Compilato con 1 errore (nessun output)]

In effetti non c'è niente di sbagliato nella punteggiatura o nell'ordine delle cose. Il problema è che abbiamo cercato di definire un oggetto con un ID interno pari a `door` [porta] - potreste pensare che sia abbastanza ragionevole, visto che l'oggetto *è* una porta – ma Inform già conosce quella parola (è il nome di un attributo di libreria). Il messaggio d'errore, sfortunatamente, fornisce solo un vago accenno al fatto che dovete scegliere un nome diverso: al suo posto abbiamo usato `toilet_door` [porta_del_bagno].

Quando il compilatore perde la traccia e non trova quello che si aspettava, è normale che le righe che seguono vengano interpretate erroneamente, anche se sono perfettamente corrette. Immaginate un metronomo che batte a tempo con un disco che state ascoltando. Se il disco è rovinato e la puntina salta, può sembrare che il resto del disco sia fuori sincrono, quando in realtà è solo un po' in ritardo a causa di quel singolo incidente. Questo succede anche con Inform, che a volte tira fuori una lista enorme di `Expected but not Found`. In questo caso la regola è: correggete il primo errore della lista e ricompilate. Potrebbe essere che il resto della canzone sia perfetta.

È inutile che vi forniamo una lista dettagliata di tutti gli errori, primo perché gli errori sono numerosi e, poi, perché di solito il testo prodotto indica quello che manca. Avete degli errori se dimenticate una virgola od un punto e virgola.

Avete degli errori se usate lo stesso nome per due cose. Avete degli errori per... un sacco di ragioni. Leggete il messaggio, andate alla riga che indica (e, magari, controllate anche quelle un po' prima ed un po' dopo), ed apportatevi quella che sembra essere la correzione più giusta.

Warning

I warning non sono immediatamente catastrofici, ma dovete liberarvene per fare una buona partenza nella ricerca degli errori di run-time [esecuzione] (leggete "Fare il debug del vostro gioco" al Capitolo 16). Potreste aver dichiarato una variabile per poi non usarla; potreste aver confuso gli operatori di assegnazione ed aritmetico (= al posto di ==); potreste aver dimenticato la virgola che separa le proprietà; ecc. ecc. Per questi warning, e per molti altri, Inform ha trovato qualcosa che è legale ma su cui ha dei dubbi.

Uno degli incidenti più comuni è quando si torna indietro (return) nel mezzo di un blocco di istruzioni, prima che il resto delle istruzioni possa essere raggiunto. Questo non sempre è evidente come si potrebbe pensare, ad esempio in un caso come questo:

```
if (steel_door has open) {
    print_ret "La brezza spegne il vostro cerino.";
    give match ~light;
}
```

Nell'esempio sopra l'istruzione `print_ret` restituisce `true` dopo che la stringa è stata stampata e, quindi, la riga `give match ~light` non verrà mai eseguita. Inform si accorge della cosa e ve lo dice. L'intenzione dello scrittore, forse, era:

```
if (steel_door has open) {
    give match ~light;
    print_ret "La brezza spegne il vostro cerino.";
}
```

La compilazione à la carte

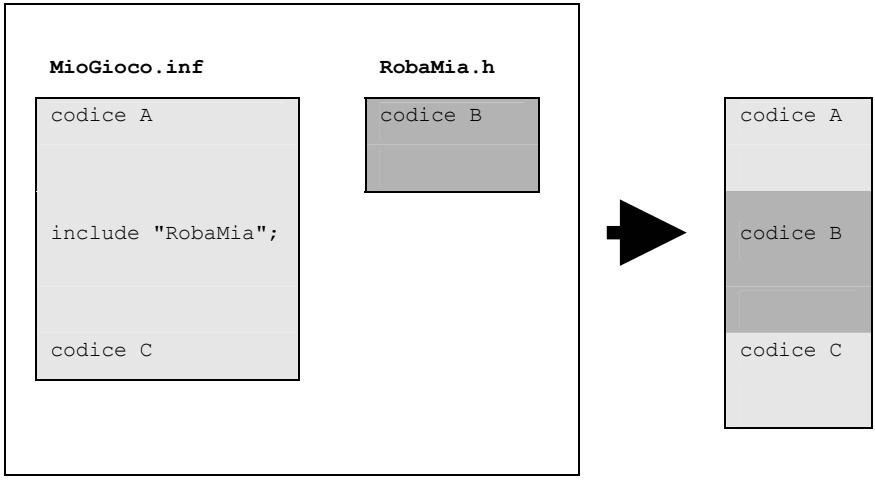
Uno dei vantaggi di Inform è la sua portabilità attraverso sistemi differenti e macchine differenti. L'utilizzo specifico del compilatore varia di conseguenza, ma alcune funzionalità dovrebbero essere presenti in tutti gli ambienti operativi. Per avere informazioni precise su di una determinata versione dovete eseguire il compilatore fornendogli lo switch `-h1`.

Il compilatore, di solito, viene eseguito solo con il nome del file sorgente come unico parametro. Questo dice al compilatore "leggi questo file e genera uno story file Versione 5 con lo stesso nome". Se vogliamo compilare un gioco in italiano, dobbiamo ricordarci d'inserire anche l'indicazione `+language_name=italian`, altrimenti il compilatore cercherà di produrre un gioco con le regole inglesi. La maggior parte del codice sorgente è piena di

istruzioni che definiscono come si debba comportare il gioco durante l'esecuzione, ma contiene anche istruzioni di compilazione indirizzate proprio al compilatore (sebbene tale comando possa sembrare un'**istruzione**, il suo comportamento è diverso, ed è conosciuta come **direttiva**). Abbiamo già visto la direttiva `Include`:

```
Include "nomefile";
```

Quando il compilatore raggiunge una riga come quella, va alla ricerca di *`nomefile`* - un altro file che contiene codice Inform - e lo tratta come se le istruzioni e le direttive contenute in nomefile si trovassero al posto della direttiva `Include`.



In ogni gioco Inform in inglese, includiamo i file di libreria `Parser`, `VerbLib` e `Grammar`. In quelli in italiano includiamo sempre `Parser`, `VerbLib`, `Replace` e `ItalianG`. Volendo possiamo includere anche altri file. Questo, ad esempio, è il modo che si usa per incorporare le estensioni della libreria scritte da altre persone, come avete visto quando abbiamo incluso `pname.h` nel nostro gioco "Capitan Fato".

> Nota: su alcune macchine, un file di libreria viene chiamato, in realtà - ad esempio - `Parser.h`, mentre su altre solo `Parser`. Il compilatore si preoccupa automaticamente di queste differenze; potete *sempre* digitare semplicemente `Include "Parser";` nel vostro codice sorgente.

Quando acquisite esperienza con Inform, e i vostri giochi diventano più complessi, potreste trovare che il file sorgente stia diventando ingestibilmente grande. Una tecnica molto utile è quella di dividerlo in sezioni, ognuna memorizzata in un file separato, che includerete in un file principale più piccolo. Ad esempio:

```
    !=============================================================
    Constant Story "Guerra e Pace";
    Constant Headline
              "^Un esempio Inform esteso
               ^by me e Leo Tolstoy.^";

    Include "Parser";
    Include "VerbLib";
    Include "Replace";

    Include "1805";
    Include "1806-11";
    Include "1812A";
    Include "1812B";
    Include "1813-20";

    Include "ItalianG";

    Include "Verboski";

!=============================================================
```

Switch (modificatori)

Quando eseguite il compilatore potete impostare alcuni controlli opzionali; questi vengono chiamati *switch* (modificatori, interruttori) e la maggior parte sono impostati (accesi) o non impostati (spenti), altri accettano un valore numerico 0-9. Gli switch modificano la compilazione in una varietà di modi, spesso cambiando semplicemente i messaggi mostrati dal compilatore quando viene eseguito. Una riga di comando tipica (sebbene possa essere differente su macchine differenti) potrebbe essere:

```
inform file_sorgente story_file switch
```

dove "`inform`" è il nome del compilatore, il parametro *`story_file`* è opzionale (in modo tale che possiate indicare un nome differente da quello di *`file_sorgente`*) ed anche gli *`switch`* sono opzionali. Un indicatore opzionale (non è propriamente uno switch) necessario se vogliamo produrre file in italiano è, come detto prima, `+language_name=italian`. Gli switch sono preceduti dal segno meno -; se volete impostare, ad esempio, il modo Strict dovete scrivere `-S`, mentre se volete disattivarlo dovete scrivere `-~S`. Il simbolo della tilde (~) deve essere interpretato, qui come altrove, come "non" ("not" in inglese). Se volete impostare molti switch potete farlo separandoli con uno spazio ed usando un meno per ogni switch, oppure raggruppandoli tutti insieme dopo un segno di meno senza mettere spazi:

```
inform MioGioco.inf -S -s -x
inform MioGioco.inf -Ssx
```

Sebbene non vi sia nulla di sbagliato in questo metodo, non è molto conveniente se dovete cambiare spesso le impostazioni degli switch. Un metodo più flessibile è quello di definirli alle primissime righe del vostro file sorgente, nelle due seguenti forme:

(1)

```
!% -SsX
```

(2)

```
!% -S
!% -s
!% -X
```

Nota la forma "`!% -S -s  -X`" non è accettata.

Tutti gli switch, normalmente, sono disattivati per impostazione predefinita, l'unica eccezione è il modo Strict (`-S`), che è attivo e controlla il codice per degli errori addizionali, rendendo disponibili i verbi di debug durante l'esecuzione del gioco. Questa è l'impostazione ideale mentre si sta scrivendo il codice, ma dovete disattivare il modo Strict (`-~S`) quando distribuite il vostro gioco. Fortunatamente è molto facile controllare la cosa, visto che il banner del gioco visualizza le lettere "`SD`" quando è stato compilato in modo Strict:

```
Capitan Fato
Un semplice esempio Inform
by Roger Firth e Sonia Kesserich.
Versione 3 / Numero di Serie 040804 / Inform v6.30 Libreria 6/11 SD
```

Gli switch fanno differenza tra maiuscole e minuscole, quindi avrete risultati differenti tra `-x` e `-X`. Alcuni degli switch più utili sono:

`-~S`

Disattiva il modo Strict del compilatore. Questo evita che vengano fatti alcuni controlli aggiuntivi al vostro codice sorgente ed, inoltre, non aggiunge i verbi di debug al vostro story file (a meno che voi non specifichiate `-D`). Il modo Strict è attivo per impostazione predefinita.

`-v5 -v8`

Compilano in quella versione di story file. Le Versioni 5 (quella predefinita) ed 8 sono le uniche di cui dovete occuparvi; producono, rispettivamente, story file con estensione `.z5` e `.z8`. La Versione 5 era quella progettata dalla Infocom come Advanced (avanzata), ed è quella che viene prodotta per impostazione predefinita da Inform. Questa è la versione che di solito utilizzerete, e consente di creare file grandi fino a 256 Kbyte. Se il vostro gioco cresce oltre quella dimensione, dovete compilare uno story file Versione 8, che è molto simile alla Versione 5, ma consente di creare file grandi fino a 512 Kbyte.

`-D -X`

Includono, rispettivamente, i verbi di debug ed il debbuger Infix nello story file (leggete "Fare il debug del vostro gioco" al Capitolo 16)

`-h1 -h2`

Visualizzano le informazioni di aiuto sul compilatore. `-h1` fornisce informazioni sulla nomenclatura dei file, mentre `-h2` visualizza gli switch disponibili.

`-n -j`

`-n` visualizza il numero di attributi, proprietà ed azioni che avete dichiarato. `-j` elenca gli oggetti così come verranno letti e costruiti nello story file.

`-s`

Mostra le statistiche sul gioco. Fornisce parecchie informazioni sul vostro gioco, compreso il numero di oggetti, verbi, voci del dizionario, uso della memoria, ecc. indicando allo stesso tempo il massimo previsto in ogni categoria. Utile per controllare se state raggiungendo i limiti di Inform.

`-r`

Memorizza tutti i testi del gioco in un file temporaneo, utile per controllare le vostre descrizioni ed i vostri messaggi attraverso un correttore ortografico.

Se eseguite il compilatore con lo switch `-h2` scoprirete che ci sono molti altri switch, che, spesso, offrono funzionalità oscure od avanzate che, crediamo, siano di scarso interesse per i principianti. Sentitevi, comunque, liberi di provare qualunque switch catturi la vostra attenzione; niente di quello che fate in questa fase modifica il vostro codice sorgente, che, per quel che riguarda il compilatore, è assolutamente sola lettura.

16. Fare il debug del vostro gioco

Nessuno comprende la frase *errare humanum est* meglio di un programmatore. I computer sono delle macchine estremamente efficienti, capaci di effettuare dei calcoli impressionanti, ma mancano d'immaginazione ed insistono sul fatto che ogni elemento venga fornito loro debba seguire una serie di regole definite in precedenza. Non potete trovare un accordo con un computer: o vi sottomettete ai suoi voleri, o mangiate la polvere.

Inform non fa differenza. Se fate un errore di battitura o di sintassi, il compilatore ve lo indica chiedendovi di correggere il vostro lavoro. "Mi sono solo scordato una virgola!" urlate con disgusto. Il compilatore non dice nulla. Non ha niente da guadagnare dalla discussione, perché ha sempre ragione. Quindi aggiungete la virgola che manca. Nessuno è rimasto ferito tranne, forse, il vostro orgoglio.

Gli errori che si trovano durante la compilazione possono essere noiosi da correggere, ma solitamente è facile individuarli; dopo tutto il compilatore cerca, gentilmente, di indicarvi quale era e dove si trovava l'errore. Il difficile viene quando avete finito di soddisfare tutte le lamentele del compilatore. Il vostro premio è uno schermo pulito, senza nessun elenco di errori, e vi viene offerto - un dono!

Un nuovo file è comparso nella vostra cartella. Uno story file. Sì, *il* gioco. Aprite, subito, il vostro interprete preferito e cominciate a giocare... solo per scoprire il lato oscuro degli errori: i bug [bachi]. I bug sono di tutte le forme, colori e dimensioni: grandi, piccoli, stupidi, assurdi, minori, inquietanti, stressanti e catastrofici. Di solito sono imprevedibili: ci intrattengono con comportamenti soprendenti ed inaspettati. Sfidano la logica: io scrivo PRENDI la chiave, ed il gioco risponde con "Presa.", ma la chiave resta allo stesso posto e non appare nel mio inventario. Oppure l'apertura di una porta avendo indosso una pelliccia causa un errore di programmazione e la visualizzazione di un messaggio criptico: "tried to find the attribute of nothing" [si è cercato di trovare l'attributo di niente]. E molti, molti altri.

Quando progettate un gioco cercate di prendere in considerazione tutti gli stati in cui i vostri oggetti potrebbero venire a trovarsi, ma qualunque gioco di medie dimensioni ha una tale quantità di oggetti ed azioni che è praticamente impossibile pensare a tutte le variazioni, permutazioni e possibilità.

Il debug [rimozione dei bachi] consiste nello scovare gli errori d'esecuzione e correggerli. Potreste pensare che sia abbastanza semplice, ma non lo è. L'individuazione di tali errori non è facile, visto che tendono a manifestarsi solo in determinate circostanze. Quindi dovete investigare nel vostro codice per scoprire che cosa li ha causati. Infine, quando avete trovato le righe incriminate, dovete apportare i cambiamenti necessari. (Esiste anche il caso in cui non

riuscite a trovare l'errore. Non vi preoccupate: è lì da qualche parte. L'insistenza alla fine paga).

Per aiutarvi in questo difficile compito, Inform ha una serie di azioni speciali: i verbi di debug [debugging verbs]. Sono disponibili durante l'esecuzione del gioco se il file sorgente è stato compilato in **modo Debug** (lo switch `-D`) od in modo Strict (lo switch `-s`, che include il modo Debug). In effetti il compilatore ha il modo Strict attivo per impostazione predefinita come una ciambella di salvataggio, visto che controlla il vostro codice più accuratamente alla ricerca di alcuni errori addizionali. Quando siete pronti a rilasciare il vostro gioco, dovete ricompilare, disabilitando il modo Strict (`-~s`) per evitare che i giocatori si avvantaggino delle funzionalità offerte dai verbi di debug. Parleremo un po' di alcuni di questi e vi faremo vedere a cosa servono.

Lista dei comandi

L'unico modo per provare (farne il test) un gioco è giocarci. Mentre andate avanti con la scrittura del codice il gioco cresce in dimensioni e diventa veramente stancante dover ripetere tutti i comandi tutte le volte che si gioca. Quando sistemate il comportamento di un certo oggetto, di solito, modificate di conseguenza il comportamento di altri oggetti od altre azioni, quindi è una buona idea fare un test generale ogni tanto: dovete assicurarvi che alcune delle modifiche più recenti non abbiano rovinato qualcosa che prima funzionava benissimo.

Il comando RECORDING [registrazione] (RECORDING ON e RECORDING OFF) memorizza i comandi che digitate mentre giocate in un file di testo (probabilmente vi verrà chiesto un nome di file). Quando aggiungete una nuova sezione al gioco, potete giocare fino a quel punto, quindi digitare RECORDING ON per catturare (in un altro file) i comandi che riguardano quella sezione, ed alla fine potete usare il vostro editor di testo per accodare i nuovi comandi all'elenco preesistente.

Il comando REPLAY [riproduci] esegue il file di testo creato da RECORDING, lanciando tutti i comandi memorizzati nel file. In questo modo potete accertarvi rapidamente se tutto sta funzionando come dovrebbe.

Potete aprire il file dei comandi con qualunque editor di testo per modificarne il contenuto a seconda delle necessità: ad esempio, se volete cancellare dei comandi non più necessari a causa di una modifica al gioco, oppure se avete dimenticato di fare il test di qualche oggetto particolare ed avete bisogno di aggiungere dei comandi.

Questa tecnica (l'uso di una lista di comandi registrati) è, e non possiamo enfatizzarlo con più forza, una delle funzionalità di testing più utili che possa avere uno scrittore di avventure.

Sputa il rospo

Alcuni verbi di debug offrono informazioni sullo stato corrente delle cose.

TREE

Questa azione elenca tutti gli oggetti nel gioco e come sono contenuti uno nell'altro. Potete scoprire il contenuto di un singolo oggetto digitando TREE *oggetto*. Tutti gli oggetti che avete definito nel vostro codice sorgente vengono trasformati in numeri da Inform quando compila lo story file; questo comando visualizza anche questa rappresentazione numerica interna degli oggetti, chiamata *obj_id*.

SHOWOBJ *oggetto*

Visualizza le informazioni che riguardano l'*oggetto*, gli attributi che possiede ed il valore delle sue proprietà. L'*oggetto* può trovarsi ovunque, non è necessario che sia nelle vicinanze (in scope). Potete anche usare il numero dell'oggetto, se l'oggetto in questione non ha un nome. Ad esempio, in "Heidi":

```
>SHOWOBJ NIDO
Object "nido di uccelli" (29) in "te stesso"
  has container moved open workflag
  with name 'nido' 'rametti' 'sterpi',
       description "Il nido e' fatto di rametti e
              sterpi intrecciati." (-28929),
```

SHOWVERB *verbo*

Visualizza la grammatica del verbo, proprio come una definizione `Verb` standard. Diventa utile quando avete giocato un po' con la funzione `Extend` e non siete sicuri del risultato finale delle vostre macchinazioni. Un esempio da "Guglielmo Tell":

```
>SHOWVERB DAI
Verb 'da' 'dai' 'offri' 'paga'
     * held   'a'/'ad'/'all^'/'allo'/'alla'/'al'/'agli'/'ai'/
              'alle' creature
                                        -> Give
     * 'a'/'ad'/'all^'/'allo'/'alla'/'al'/'agli'/'ai'/'alle'
       creature held
                                        -> Give reverse
     * 'omaggio'/'omaggi' 'a'/'ad'/'all^'/'allo'/'alla'/'al'/
       'agli'/'ai'/'alle'/'verso' noun
                                        -> Salute
```

Le prime righe riproducono la definizione del verbo così come si trova nella libreria. L'ultima riga, invece, è la diretta conseguenza del nostro `Extend`:

```
Extend 'dai'
    * 'omaggio'/'omaggi' 'a'/'ad'/'all^'/'allo'/
      'alla'/'al'/'agli'/'ai'/'alle'/'verso' noun
                                        -> Salute;
```

SCOPE

Elenca tutti gli oggetti attualemente *in scope* (in parole povere, visibili al personaggio del giocatore). In maniera più potente potete digitare SCOPE *oggetto* per scoprire quali sono gli oggetti *in scope* per l'oggetto indicato. Questa funzionalità diventa utile soprattutto quando avete degli NPC in grado di interagire con l'ambiente circostante.

Che diavolo sta succedendo?

Ci sono momenti in cui una determinata azione non produce l'effetto che vi aspettavate e non sapete perché. I verbi di debug che seguono offrono informazioni su quello che l'interprete sta facendo, e potrebbero aiutarvi a capire in quale momento le cose hanno smesso di funzionare.

ACTIONS (o ACTIONS ON) e ACTIONS OFF

Fornisce informazioni sulle azioni che sono in corso. Alcune azioni vengono reindirizzate ad altre, e questo può essere una fonte di guai o di mistero; in questo modo avete un'indicazione di quello che sta succedendo. Ad esempio, date un'occhiata a questa trascrizione da "Guglielmo Tell":

```
Lungo la strada
La gente continua a premere e a farsi strada dalla porta sud alla
piazza principale, che si trova appena più a nord. Riconosci la
proprietaria di un banco di frutta e verdura.
Helga smette di sistemare le patate e ti saluta calorosamente.

>CERCA BANCO
[ Action Search with noun 35 (banco di frutta e verdura) ]
[ Action Examine with noun 35 (banco di frutta e verdura) (from < >
statement) ]
Davvero un piccolo banco, con un grosso mucchio di patate, qualche
carota, qualche rapa, un po' di mele.
...
```

CHANGES (o CHANGES ON) e CHANGES OFF

Tiene traccia dei movimenti degli oggetti e delle modifiche agli attributi ed alle proprietà:

```
In mezzo alla piazza
C'è meno folla al centro della piazza; la maggior parte della gente
preferisce tenersi il più lontano possibile dal palo che troneggia
in questo luogo, reggendo quell'assurdo cappello cerimoniale. Un
gruppo di soldati rimane nei pressi, osservando chiunque passi di
qui.

>NORD
[Setting In mezzo alla piazza.avvertimenti to 1]

Un soldato ti sbarra la strada.

"Hey, tu, spilungone; hai dimenticato le buone maniere? Forse
sarebbe il caso di fare un bel saluto al cappello del balivo, non
trovi?"
```

```
>ANCORA
[Setting In mezzo alla piazza.avvertimenti to 2]
"Ti conosco, Tell, sei uno che porta solo guai, vero? Non vogliamo
teste calde qui, quindi fai il bravo ragazzo e porgi il tuo saluto
al dannato cappello. Fallo ora, non voglio chiedertelo di nuovo..."

>SALUTA CAPPELLO
[Setting palo di legno.salutato to 1]
Saluti il cappello sull'alto palo.

"Grazie davvero, messere", sghignazzano i soldati.

>SUD
[Setting In mezzo alla piazza.avvertimenti to 0]
[Setting palo di legno.salutato to 0]
[Moving te stesso to Lato sud della piazza]
...
```

TIMERS (o TIMERS ON) e TIMERS OFF

Questo verbo visualizza, alla fine di ogni turno, lo stato di tutti i `timer` e dei `daemon` attivi. Non abbiamo parlato dei timer (simili ai daemon) in questa guida; potreste usarne uno per far esplodere una bomba dieci turni dopo aver acceso la sua miccia.

TRACE (o TRACE ON), TRACE *numero* e TRACE OFF

Quando attivate questo verbo (molto potente) siete in grado di seguire l'attività del parser (la parte della libreria che cerca di capire quello che il giocatore ha digitato) e questo sarà un momento di estrema gratitudine per il fatto che qualcun altro si è preso l'onere di scriverlo. Visto che il parser fa tantissime cose, potete decidere il livello di dettaglio delle informazioni mostrate tramite il parametro *numero*, che può variare tra 1 (informazioni minime) e 5 (informazioni massime). Per impostazione predefinita TRACE ON e TRACE impostano il livello a 1. Il livello 1 mostra la riga di grammatica sulla quale il parser sta pensando, mentre il livello 2 mostra ogni singola chiave di ogni riga di grammatica che prova. Le informazioni visualizzate ai livelli più alti cominciano ad essere abbastanza intricate, e vi consigliamo di usarle solo se nessun'altra cosa vi è stata di aiuto.

Superpoteri

GONEAR *oggetto*

Questa azione vi teletrasporta nella stanza in cui si trova l'*oggetto*. Diventa utile quando, ad esempio, talune parti della mappa sono chiuse fino a che il giocatore non risolve qualche enigma, o se la mappa del gioco è divisa in aree differenti. Se la stanza che volete visitare non ha oggetti, potete usare...

GOTO *numero*

Vi teletrasporta nella stanza che ha come rappresentazione interna *numero*. Visto che le stanze, di solito, non hanno nome, dovete scoprire il numero interno dell'oggetto stanza (ad esempio con il comando TREE).

PURLOIN *oggetto*

PURLOIN funziona proprio come PRENDI, con la simpatica aggiunta che non si cura di dove si trovi l'*oggetto*: in un'altra stanza, all'interno di un contenitore chiuso a chiave, negli artigli del drago assetato di sangue. In maniera più pericolosa, non si cura neanche di controllare che un oggetto sia prendibile, quindi potete prendere degli oggetti definiti come `static` o `scenery`. PURLOIN è utile in una varietà di situazioni, fondamentalmente quando volete provare una certa funzionalità del programma che richiede al giocatore di avere determinati oggetti a portata di mano. Invece di andare in giro a raccoglierli, potete semplicemente impossesarvene con il comando PURLOIN. Fate attenzione: non è saggio impossessarsi di oggetti che non è previsto siano prendibili, visto che in questo modo il comportamento del gioco non sarà prevedibile.

ABSTRACT *oggetto* TO *oggetto*

Questo verbo vi consente di muovere il primo *oggetto* nel secondo *oggetto*. Così come con PURLOIN, gli oggetti possono trovarsi ovunque nel gioco. Tenete a mente che il secondo oggetto deve logicamente essere un `container`, un `supporter` o qualcosa di animato (`animate`).

Infix: il privilegio del corrotto

I verbi base di debug sono abbastanza versatili ed hanno il vantaggio di essere compilati automaticamente all'interno del vostro gioco a meno che non lo vogliate. A volte, comunque, potreste incontrare un bug che non riuscite a stanare usando le tecniche normali, ed è questo il momento in cui vorrete dare un'occhiata al debugger Infix. Dovete compilare usando lo switch `-X` per essere, così, in grado di modificare quasi tutti i dati e gli oggetti del vostro gioco. Ad esempio potete usare "`;`" per leggere e modificare il contenuto di una variabile:

```
All'interno del caffè di Benny
Da Benny potete trovare le migliori paste ed i migliori sandwich. I
clienti si ammassano al bancone, dove lo stesso Benny si occupa di
servire, cucinare e dare il resto senza perdere un colpo. Nella
```

```
parte nord del bar potete vedere una porta rossa che conduce al
gabinetto.

>; deadflag

; == 0

>; deadflag = 4

; == 4

*** Sei stato VERGOGNOSAMENTE sconfitto ***

In questa partita hai totalizzato 0 punti su 2 possibili, in 2
turni.
```

Spesso è abbastanza scocciante scoprire che una variabile è ancora `false` perché l'enigma del Gessetto non ha funzionato e che non potete verificare l'enigma del Formaggio fino a che non diventa `true`. Invece di uscire dal gioco, sistemare il gessetto, ricompilare, giocare fino alla posizione che avete lasciato e *solo allora* affrontare il formaggio, sarebbe molto più facile cambiare il valore della variabile a metà strada e proseguire.

Potete usare `;WATCH` per tenere d'occhio un oggetto, lo vedrete ricevere messaggi e vi sarà segnalato quando i valori delle sue proprietà e attributi cambieranno.:

```
>;WATCH CENTRO_PIAZZA
; Watching object "In mezzo alla piazza" (43).


>NORD
[Moving te stesso to In mezzo alla piazza]
[Moving gente to In mezzo alla piazza]
[Moving soldati di Gessler to In mezzo alla piazza]
[Moving tuo figlio to In mezzo alla piazza]


In mezzo alla piazza
C'è meno folla al centro della piazza; la maggior parte della gente
preferisce tenersi il più lontano possibile dal palo che troneggia
in questo luogo, reggendo quell'assurdo cappello cerimoniale. Un
gruppo di soldati rimane nei pressi, osservando chiunque passi di
qui.

[Giving In mezzo alla piazza visited]


>NORD
[ "In mezzo alla piazza".before() ]
[ centro_piazza.before() ]
[Setting In mezzo alla piazza.avvertimenti to 1]
Un soldato ti sbarra la strada.

"Hey, tu, spilungone; hai dimenticato le buone maniere? Forse
sarebbe il caso di fare un bel saluto al cappello del balivo, non
trovi?"


>NORD
[ "In mezzo alla piazza".before() ]
[ centro_piazza.before() ]
[Setting In mezzo alla piazza.avvertimenti to 2]
```

```
"Ti conosco, Tell, sei uno che porta solo guai, vero? Non vogliamo
teste calde qui, quindi fai il bravo ragazzo e porgi il tuo saluto
al dannato cappello. Fallo ora, non voglio chiedertelo di nuovo..."


>NORD
[ "In mezzo alla piazza".before() ]
[ centro_piazza.before() ]
[Setting In mezzo alla piazza.avvertimenti to 3]
"Va bene, Herr Tell, ora siete nei guai.
...
```

Infix è abbastanza complesso; è utile averlo a disposizione, ma non è decisamente uno strumento per principianti. Se proprio volete usarlo fate attenzione: avrete un sacco di potere e sarà vostra cura non rovinare tutto. Non ne avrete bisogno spesso, ma Infix può fornire risposte rapide a questioni intricate.

Non importa cosa

Il vostro gioco avrà ancora dei bug irrisolti nonostante tutti i vostri sforzi per sistemarlo. È una cosa normale, anche per gli scrittori di avventure che hanno una certa esperienza: non vi scoraggiate o demoralizzate. Potreste trovare rassicurante sapere che anche i nostri stessi giochi di esempio in questa guida – che di certo non possono essere qualificati come "programmazione complessa" – erano ben lontani dall'essere perfetti nella Prima Edizione. Arrossimmo di vergogna quando ricevemmo questo resoconto da un playtester estremamente diligente:

> Ho trovato queste cose giocando a "Capitan Fate":
>
> - il giocatore può indossare gli abiti sopra il costume,
> - il giocatore può indossare il costume al buio nel bagno non bloccato senza essere interrotto,
> - il giocatore può lasciare i vestiti nel bagno se questo è al buio e con la porta non bloccata. Provare TOGLI I VESTITI. X ME. TOGLI IL COSTUME. INV – X ME dice che si stà indossando il costume, ma l'inventario non corrisponde a tale situazione.

La Seconda Edizione di questa guida ha posto rimedio a tali problemi, ed anche a qualcun altro. "È fatta", abbiamo pensato, "dopo tutto questo tempo, i nostri giochi sono sicuramente perfetti". Nei nostri sogni… Un altro diligente playtester in seguito ci scrisse:

> Mentre leggevo, ho preso nota di alcuni errori ed incoerenze:
>
> - i comandi BENNY, DAI CHIAVE AGLI AVVENTORI e BENNY, DAI LA CHIAVE hanno l'effetto di dare la chiave al giocatore. Lo stesso accade per il caffè.

- Benny costringerà il giocatore a tornare indietro nel bar anche se la chiave viene lasciata nel bar, o messa sul bancone (in piena vista!).

Naturalmente, il codice che vi abbiamo offerto in questa Terza Edizione ha posto rimedio a queste imbarazzanti segnalazioni, ma potrà facilmente accadere che altri piccoli difetti od incoerenze saltino fuori più avanti.

La fase finale del processo di debug deve avvenire da un'altra parte, nelle mani di alcuni beta-tester volenterosi, testardi e determinati; sono queste le persone che, se siete fortunati, faranno a pezzi il vostro gioco e vi restituiranno estesi rapporti in cui vi dicono le cose che non funzionano in maniera corretta, le cose che non funzionano così bene come dovrebbero, le cose che avrebbero dovuto funzionare ma non lo fanno, e le cose che non vi sono neanche venute in mente (come, uh, lasciare il costume al buio). Quando credete che il vostro gioco sia finito (nel senso che fa tutto quello che pensate avrebbe dovuto fare, e non avete più idee su come fare altri test) cercate qualche beta-tester: un buon numero è tre o quattro. La comunità IF offre alcune risorse per il beta-testing, oppure potete sempre chiedere su RAIF (in inglese) o su ICGAT (in italiano) e cercare qualche anima pia che si prenda la briga di dare un'occhiata al vostro gioco. Ricordatevi sempre le regole d'oro:

- Non vi aspettate nessuna pietà. Sebbene vi possa ferire, un approccio senza pietà è quello di cui avete bisogno in questo momento; è molto meglio scoprire i vostri errori e le vostre dimenticanze adesso, piuttosto che quando rilasciate il gioco al pubblico. E non dimenticate di ringraziare i vostri beta-tester da qualche parte nel gioco.
- Mai dire mai. Se i vostri tester vi dicono che il gioco dovrebbe rispondere meglio ad una certa azione che non avete provato, non rispondete automaticamente con "Nessuno lo farà mai!". Lo hanno appena fatto, e qualcun altro lo farà: siate grati alle menti deviate dei vostri tester. Anche se un giocatore normale non proverà mai *tutte* quelle cose strane, ogni giocatore sarà propenso a provarne *almeno* una, ed il loro divertimento sarà più grande, la sensazione di realtà sarà migliore, se il gioco "capisce".
- Chiedete di più. Non trattate i vostri tester come dei semplici validatori delle vostre capacità programmatorie, ma piuttosto come dei recensori delle vostre abilità di scrittore. Incoraggiateli a commentare quanto bene i pezzi si uniscono l'un l'altro, ed a fornire suggerimenti (piccoli o radicali) per migliorare; non è necessario rifiutare una buona idea solo perché "ci vuole troppo per implementarla". Ad esempio: "la scena nella Torre di Londra non sembra accordarsi molto con un gioco sulle Notti Arabe", oppure "dover risolvere tre enigmi uno dopo l'altro solo per scoprire un piatto di occhi di pecora è un po' sopra le righe", o "questo viaggio di cinque stanze nel deserto è un po' noioso: forse potresti aggiungere delle sabbie mobili o qualcos'altro per rinvigorirlo?", o "sembra che al personaggio dell'eunuco nell'harem manchi qualcosa". Quindi cercate di vedere l'insieme dei vostri

beta-tester non come dei semplici correttori ortografici, ma come dei veri e propri redattori che collaborano alla scrittura del vostro romanzo.

17. *** Hai vinto ***

I might just as well have saved the labor and sweat I had put into trying to make my reports harmless. They didn't fool the Old Man. He gave me merry hell.

- The Continental Op in Dashiel Hammett's Red Harvest[16].

Solo qualche parola finale per chiudere il cerchio. Quello che resta sono le appendici, con un sommario conciso ma completo del linguaggio Inform e della sua libreria IF, ed il codice sorgente e la trascrizione dell'esecuzione dei giochi che abbiamo sviluppato insieme. Il nostro "lavoro e sudore" era orientato a rendere la vostra introduzione ad Inform il più indolore possibile, ma forse non vi inganneremo a lungo. Sebbene siamo convinti di aver coperto le funzionalità base del sistema e di avervi dato una base abbastanza solida per affrontare le insidie della progettazione di avventure testuali, esistono ancora molte tecniche da padroneggiare ed ulteriori aspetti da apprendere, comprese le funzionalità medie ed avanzate che non abbiamo neanche accennato.

Prima che ci provochiate un sacco di guai, comunque, vi possiamo assicurare che le conoscenze rimanenti, che potrebbero sembrare oscure ed enigmatiche, sono costruite, fondamentalmente, intorno agli stessi principi che avete già visto. Dovreste essere in grado di sfogliare dell'altra documentazione e delle altre risorse senza trovare che siano piene di strani geroglifici; al contrario sarete in grado di focalizzare la vostra attenzione su quelle parti che non conoscete (che ora, speriamo, saranno meno numerose). Inform, come altri sistemi di scrittura di IF potenti e flessibili, è in grado di far fronte alle necessità degli autori più esigenti: "Non mi piace il modo in cui PRENDI TUTTO è gestito: lo voglio cambiare!" E potete farlo. "Mi piacerebbe che l'elenco degli oggetti venisse organizzato in un modo migliore." Fatelo. "Voglio avere una migliore vita sociale grazie ad Inform." Nessun problema, ma dovete essere uno scrittore maledettamente affascinante. Oh, vabbè.

Inform è stato progettato per farvi fare le cose semplici in maniera intuitiva e veloce. Lasciato così com'è vi offre un ampio spettro di funzionalità predefinite, ed abbiamo anche visto che è facile modificare alcuni dei comportamenti standard. L'obiettivo è quello che raggiungiate un tale livello di familiarità con il sistema in modo che possiate concentrarvi sulla progettazione del vostro gioco. Con "livello di familiarità" non intendiamo che voi conosciate tutti i segreti della

[16] Potevo anche aver salvato il lavoro ed il sudore che avevo impiegato nel tentare di rendere i miei rapporti inoffensivi. Non avevano ingannato il Vecchio Uomo. Mi ha provocato un sacco di guai.

libreria; queste persone esistono, ma sono poche. Tuttavia, una volta che siete diventati ragionevolmente esperti nella scrittura del codice, con un livello di conoscenza simile a quello fornito da questa guida, un'occhiata alla sezione appropriata dell'*Inform Designer's Manual* dovrebbe aiutarvi quando siete in difficoltà. È anche vero che ci sono problemi e problemi, dalla stupidaggine da pacca sulla fronte, all'incubo notturno da far digrignare i denti. Vi consigliamo di posticipare gli incubi. Potrebbe essere che un giorno scopriate che le loro zanne non sono così affilate.

Ci sono molti argomenti interessanti che potreste affrontare in seguito. Eccone alcuni:

- **Punti**: abbiamo visto due modi di fare punti in un gioco, ma potreste decidere che il punteggio non abbia significato nel vostro gioco. E c'è un terzo modo incorporato nel sistema per definire quali sono i "compiti" che meritano una ricompensa, dallo "indossare una ridicola cuffia alla festa dell'Ambasciatore" al "convincere la poco amichevole scimmia a suonare il pianoforte". Queste tecniche richiedono un po' di conoscenza di...
- **Array** (matrici): sono liste numerate di variabili. Invece di avere un'unica variabile con cui giocare, potete averne un insieme, indicizzato da un numero.
- **Liste ed inventari**: ci sono molte funzioni che vi consentono di modificare il modo in cui gli oggetti vengono raggruppati e presentati al giocatore durante l'esecuzione del programma.
- **Veicoli**: auto, ascensori, mongolfiere, tappeti magici, navi spaziali - o qualunque altro dispositivo consenta al giocatore di muoversi.
- **La creazione di verbi e del vocabolario**: sebbene abbiamo già affrontato questo concetto, potete regolare il parser per fargli fare qualunque tipo di cose speciali (da pronunce magiche che scatenino incantesimi impossibili, ad azioni speciali che riguardino più oggetti insieme).
- **Cambiare il giocatore**: chi l'ha detto che il personaggio del giocatore debba essere un noioso essere umano? Trasformate l'insospettabile mortale in un proxy della realtà virtuale, un animale fantastico, un fantasma immateriale, un potente telepate od un vampiro telecinetico. Non sapete quale scegliere? Allora il vostro gioco può avere più personaggi principali e potete scegliere con chi giocare.
- **Lo scorrere del tempo, macchine a tempo ed eventi**: impostate un timer che segni il tempo, sconosciuto al giocatore ed attaccatelo ad una bomba; una porta che si apre solo una volta ogni dieci turni; un drago di poche parole e poca pazienza; una pattuglia di soldati; un orologio che, inquietante, annunci l'arrivo del tramonto e del destino. Cambiate le "Azioni" sulla riga di stato in minuti, od in giorni.

- **Direzioni che cambiano**: a nord c'è il nord? Non necessariamente. Cambiate gli oggetti di direzione del gioco in "avanti", "indietro" e così via. Siete su una nave? "Prua" e "poppa", "babordo" e "dritta" possono fare per voi. Entrate in uno specchio ed avrete la mappa e tutte le direzioni riflesse.
- **NPC complessi**: quanto imprevedibile è il comportamento di quell'impertinente del maggiordomo? Può parlare, muoversi, rubarvi la roba, avvelenare il vostro thè? Reagisce in maniera coerente alle azioni del giocatore? Tiene una sua agenda? Sebbene la creazione di un personaggio non giocatore è un bel nodo da sciogliere, è una cosa che vale la pena padroneggiare. NPC "vivi" aumentano tantissimo la sensazione di realtà del vostro gioco.
- **Funzionalità tecniche**: cambiate la riga di stato, od il prompt dei comandi. Pulite lo schermo, o modificatene il colore; centrate il testo sullo schermo, e cambiate il colore anche di quello. Aspettate che il giocatore prema un tasto e quindi eseguite una qualche azione. Visualizzate un messagio un carattere alla volta. Aggiungete una piccola bussola che mostri le uscite disponibili.

L'interactive fiction unisce la creatività e le capacità narrative con l'abilità nello scrivere codice. Di solito i giochi che colpiscono di più sono quelli che hanno una buona quantità di entrambi. Se pensate di essere carenti in una di queste qualità potete prendere in considerazione l'idea di un lavoro di gruppo: ci sono liste di collaborazione su Internet (le IF collaboration lists) dove le persone si offrono di dare una mano con idee e programmazione (e alcuni ottimi giochi sono usciti dagli sforzi congiunti di una collaborazione ben gestita). Soprattutto non dimenticate l'importanza del beta-testing, che può produrre il ritorno necessario a farvi trasformare il vostro decente tentativo in un capolavoro. Non c'è niente di meno piacevole per un giocatore che trovare un gioco che non è stato provato bene. Far fuori i bug è una vostra responsabilità; assicuratevi di fare pulizia al meglio che potete, ma non rilasciate *mai* un gioco che non è stato provato da qualcun altro. E ricordate che i beta-tester sono (quasi certamente) giocatori con esperienza, quindi i loro suggerimenti (al di là della ricerca dei bug) sono dei consigli preziosissimi di cui dovete tener conto. Incoraggiateli a scrivere commenti *sia* riguardo la programmazione *che* il progetto del gioco.

Adesso: dove andare, che fare? Consentiteci di insistere ancora una volta sull'importanza di leggere l'*Inform Designer's Manual*, un libro eccellente sotto tutti gli aspetti[17]. E visto che ci siete, scrivete dei piccoli giochi, esercizi di allenamento; non vi consigliamo di cominciare con una saga epica come vostro

[17] Il DM4 non è ancora stato tradotto in italiano, potreste trovare ostico leggerlo cio non di meno ci sentiamo di incoraggiarvi alla sua lettura. Esso contiene, magari nascoste tra le note, molte delle soluzioni che vi affannate a cercare. Un'altra buona lettura con cui proseguire il vostro apprendimento può essere il manuale di Vincenzo Scarpa. In particolare i capitoli dedicati alla traduzione di Ruins (il gioco di esempio e filo conduttore del DM4) in italiano. NdT.

primo scenario, ma se non vi va bene nient'altro – l'approccio Penso In Grande – non saremo noi a fermarvi. È una buona idea giocare con i lavori di altre persone, perché in questo modo conoscerete il livello medio che i giocatori si aspettano; controllate i newsgroup per avere suggerimenti su dei buoni titoli da provare. Intorno a Settembre assicuratevi di tenere un occhio aperto sull'Interactive Fiction Competition (`http://www.ifcomp.org`), una vetrina annuale per giochi (più o meno) corti. Leggete la fanzine Terra d'IF che spesso propone articoli dedicati alla progettazione di giochi di avventura e alla programmazione in Inform.

E, poi, chi lo sa? Magari l'anno prossimo saremo tutti colpiti dal vostro lavoro.

Sonja e Roger.

Appendice A: Come si gioca con una AT

Giocare con un'IF non richiede una gran quantità di informazioni: poche linee guida generali sono sufficienti per cominciare. Tutto quello che dovete fare è leggere le descrizioni e le situazioni che si presentano sullo schermo e poi dire al gioco quello che vorreste che accada. Immaginate di dover dare degli ordini al computer, dandogli del tu: qualcosa tipo "COMPUTER..." e l'ordine che avete pensato. In realtà non dovete scrivere COMPUTER, ma dovete scrivere l'ordine che segue, in modo da istruire il gioco ad agire al vostro posto. Di solito i comandi prendono la forma di semplici frasi imperative, con un verbo ed un complemento oggetto (ad esempio, scrivendo ESAMINA IL BOLLITORE si avrà la descrizione del bollitore, mentre con PRENDI IL BOLLITORE questo verrà a far parte delle cose in vostro possesso, e così via). Se c'è più di un bollitore nelle vicinanze, è necessario essere specifici (PRENDI IL BOLLITORE ROSSO); altrimenti il gioco potrebbe rispondere con qualcosa tipo: "Cosa intendi, il bollitore rosso oppure il bollitore arrugginito?" In questo caso è sufficiente rispondere ROSSO. Alcuni comandi possono riferirsi a due oggetti: un esempio tipico è METTI IL BOLLITORE SUL TAVOLO.

Mettiamo le parole che digitate in maiuscolo solo per evidenziarle sulla pagina. In realtà potete usare sia le lettere maiuscole che quelle minuscole, non fa differenza, e potete anche omettere di scrivere parole come IL (sebbene TAGLIA LA CORDA e TAGLIA UNA CORDA [oppure GETTA LA SPUGNA e GETTA UNA SPUGNA] possano avere effetti differenti, così come PRENDI UNA MONETA e PRENDI LA MONETA se ce ne sono diverse tra cui scegliere).

Per muoversi si usa il verbo VAI seguito da uno dei punti cardinali: VAI NORD (o VAI A NORD) ci porterà nella direzione desiderata. Di solito ci si muove parecchio, quindi è possibile scrivere semplicemente NORD, e si possono usare anche le iniziali della direzione nella quale ci vogliamo muovere (più facile e più veloce da digitare): N, S, E, O (oppure W), NE, NO (oppure NW), SE e SO (oppure SW). Ci sono anche SU (U), GIÙ (D) e, qualche volta, DENTRO e FUORI.

C'è un nutrito insieme di azioni standard sulle quali fare affidamento per ottenere qualche risultato, anche se è solo quello di farvi sapere che state perdendo del tempo. D'altra parte non dovete giocare con un'IF con una lista come questa davanti a voi, visto che l'idea sarebbe quella che un buon gioco dovrebbe essere in grado di capire qualunque cosa sembri logica da fare in una certa situazione. A volte può essere un'azione standard, altre un verbo particolare come OMAGGIA o FOTOGRAFA che, sebbene meno comuni, hanno perfettamente senso nel contesto in cui si trovano:

ACCENDI	CHIUDI	LEGA	SBLOCCA
AGITA	COMPRA	LEGGI	SCAVA
ANNUSA	DAI	MANGIA	SIEDI
APRI	DORMI	METTI	SPEGNI
ARRAMPICATI	ENTRA	MOSTRA	SPINGI
ASCOLTA	ESAMINA	NUOTA	SVUOTA
ASPETTA	ESCI	PARLA	TAGLIA
ASSAGGIA	GIRA	PENSA	TIRA
BACIA	GUARDA	PREGA	TOCCA
BEVI	INDOSSA	PRENDI	TRASFERISCI
BRUCIA	INSERISCI	PULISCI	UCCIDI
CANTA	INVENTARIO	RIEMPI	VAI
CERCA	LANCIA	RIMUOVI	
CHIEDI	LASCIA	SALTA	

Scoprirete che molte di queste sono spesso irrilevanti. Provate prima le cose logiche (se avete una torcia, BRUCIA può essere promettente, mentre MANGIA forse no). Particolarmente utili sono GUARDA (abbreviato in G o L) per visualizzare una descrizione della locazione in cui vi trovate; ESAMINA (o X) *oggetto*, che fornisce una descrizione dettagliata dell'oggetto; INVENTARIO (INV o I) per elencare gli oggetti che possedete.

Potete combinare alcuni di questi verbi con delle preposizioni per espanderne le possibilità: GUARDA ATTRAVERSO, GUARDA NEL, GUARDA SOTTO eseguono tutte azioni differenti. Ricordatevi, comunque, che stiamo menzionando solo una parte dei possibili verbi; se pensate che qualcos'altro debba funzionare, provate e vedete che succede.

Potete cambiare il modo in cui il gioco vi mostra le descrizioni delle locazioni quando ci arrivate. Il comportamento predefinito, di solito, è NORMALE (BRIEF), che vi fornisce delle descrizioni estese solo la prima volta che entrate in una determinata locazione. Alcune persone preferiscono cambiarlo in LUNGO (VERBOSE), che vi fornisce *sempre* la descrizione estesa.

Ecco una lista di altri comandi speciali, e delle relative abbreviazioni, che dovete conoscere:

ANCORA ripete l'azione che avete appena fatto.

ASPETTA (Z) salta un turno d'azione mentre voi aspettate per vedere quello che succede.

FINE (QUIT o Q) termina il gioco.

SALVA (SAVE) salva la posizione corrente nel gioco.

CARICA (RESTORE) ricarica una posizione salvata precedentemente.

RICOMINCIA (RESTART) comincia di nuovo dall'inizio.

PUNTEGGIO vi dice a che punto siete.

ANNULLA (A o UNDO) torna indietro di un turno, come se l'ultima vostra azione non sia mai stata fatta.

Spesso ci saranno dei personaggi con i quali interagire. Supponiamo che incontriate vostra cugina Maria: potreste scrivere CHIEDI A MARIA CIRCA *qualcosa*, DAI (o MOSTRA) un *oggetto* A MARIA, o CHIEDI A MARIA l'*oggetto*. I personaggi potrebbero avere voglia di aiutarvi quando esprimete le vostre richieste con: MARIA, VAI A NORD o MARIA, PRENDI LA PISTOLA. Se volete veramente bene a Maria, potreste provare a scrivere BACIA MARIA, e se vi fa inquietare oltre ogni misura potreste scrivere COLPISCI MARIA.

Una volta che vi siete riferiti ad un oggetto o ad un personaggio per nome, potete usare i pronomi ESSO, LUI, LEI (o le forme verbali contratte come PRENDILO, BACIALA) per risparmiare qualche battuta. Questi pronomi restano impostati finché non vi riferite ad un altro oggetto o ad un altro personaggio. Se volete controllare a cosa sono assegnati i pronomi, potete farlo con il comando PRONOMI.

È buona abitudine che cerchiate di mantenere le vostre azioni semplici. Molti giochi sono in grado di capire comandi lunghi come PRENDI TUTTO DALLA BORSA ECCETTO LA PERLA VERDE QUINDI TIRA IL FORMAGGIO CAMEMBERT ALLA BRUTTA MATRONA, ma queste cose sono difficili da scrivere senza fare errori di battitura. Troverete, anche, che alcune cose non sempre funzionano: TORNA IN CUCINA o AVVICINATI AL PIRATA CANTERINO o LEGGI IL GIORNALE DA SOPRA LA SPALLA DELLO SCERIFFO, potrebbero darvi errori di qualche tipo.

Capire la convenzione con cui si inseriscono i comandi è abbastanza intuitivo e ne sarete padroni dopo un po' di allenamento.

> NOTA: stiamo parlando delle funzionalità base che la maggior parte dei giochi Inform fornisce (sebbene molto sia applicabile anche ad altri sistemi di creazione di IF). Spesso gli scrittori estendono queste funzionalità definendo dei comandi addizionali, adatti al tipo di gioco; nel caso vi verrà detto, o vi verranno naturalmente in mente nel corso del gioco. Meno frequentemente, alcuni scrittori si divertono a cambiare il comportamento standard del parser (l'analizzatore lessicale), a cambiare l'interfaccia, od il modo in cui i comandi lavorano: disabilitando, a volte, alcuni dei comandi standard. Quando capita, è pratica comune e rispettosa del gioco farvelo sapere.

Appendice B: la storia di "Heidi"

Heidi nella Foresta è il nostro primo – e più semplice – gioco. L'abbiamo descritto in tre capitoli: "Heidi: il nostro primo gioco in Inform" nel terzo capitolo, "Riepiloghiamo le basi" nel quarto capitolo e "Rivediamo Heidi" nel quinto capitolo. Qui riportiamo la trascrizione di una partita, e quindi il file sorgente originale in tutta la sua estensione .

Trascrizione di una partita

```
Heidi
Semplice esempio in Inform
di Roger Firth e Sonja Kesserich.
Versione 1 -- Numero di serie 061108
Inform v6.31 -- Libreria 6/11  --  Infit v2.5    SD

Di fronte a una baita
Ti trovi davanti a una baita. Verso est si stende la foresta.

>LUNGO
Heidi è ora in modalità "completa", che dà descrizioni lunghe per tutti
i luoghi (anche se già visitati).

>VAI A EST

Nel folto del bosco
Attraverso la folta vegetazione, ti pare di scorgere un edificio verso
ovest. Un sentiero porta verso nord-est.

Puoi vedere un uccellino qui.

>ESAMINA L'UCCELLINO
Troppo giovane per saper volare, il passerotto pigola inerme.

>PRENDILO
Preso.

>NE


Una radura nella foresta
Un alto sicomoro si erge al centro di questa radura. Il sentiero si
inoltra tra gli alberi, verso sud-ovest.

Puoi vedere un nido di uccelli (che è vuoto) qui.

>METTI L'UCCELLINO NEL NIDO
Hai messo l'uccellino dentro il nido di uccelli.

>ESAMINA IL NIDO
Il nido è fatto di rametti e sterpi intrecciati.

>PRENDI IL NIDO
Preso.
```

```
>SU

In cima all'albero
Ti tieni precariamente appesa al tronco.

Puoi vedere un ramo largo e robusto qui.

>METTI IL NIDO SUL RAMO
Hai messo il nido di uccelli sopra il ramo largo e robusto.


    *** Hai vinto ***

In questa partita hai totalizzato 0 punti su 0 possibili, in 9 turni.
Vuoi RICOMINCIARE, CARICARE una partita salvata o USCIRE ?
>USCIRE
```

Codice sorgente del gioco - versione originale

```
!% -SD
!=============================================================================
Constant Story "Heidi";
Constant Headline
            "^Semplice esempio in Inform
             ^di Roger Firth e Sonja Kesserich.^";
             ! Traduzione di Paolo Lucchesi
Constant MAX_CARRIED 1;

Include "Parser";
Include "VerbLib";
Include "Replace";

!=============================================================================
! Oggetti di gioco


Object  davanti_baita "Di fronte a una baita"
    with
        description
            "Ti trovi davanti a una baita. Verso est si stende la
             foresta.",
        e_to foresta,
    has light;

Object  foresta "Nel folto del bosco"
    with
        description
            "Attraverso la folta vegetazione, ti pare di scorgere un
             edificio verso ovest. Un sentiero porta verso nord-est.",
        w_to davanti_baita,
        ne_to radura,
    has light;


Object  uccello "uccellino" foresta
```

```
    with
        description
            "Troppo giovane per saper volare, il passerotto pigola
             inerme.",
        name 'uccello' 'uccellino' 'passero' 'passerotto' 'volatile',
    has ;

Object  radura "Una radura nella foresta"
    with
        description
            "Un alto sicomoro si erge al centro di questa radura. Il
             sentiero si inoltra tra gli alberi, verso sud-ovest.",
        sw_to foresta,
        u_to sull_albero,
    has light;

Object  nido "nido di uccelli" radura
    with
        description
            "Il nido @`e fatto di rametti e sterpi intrecciati.",
        name 'nido' 'rametti' 'sterpi',
    has container open;

Object  albero "albero di sicomoro" radura
    with
        description
            "Fieramente alto nel mezzo della radura, l'albero sembra
             molto facile da scalare.",
        name 'albero' 'sicomoro' 'tronco',
    has scenery;

Object  sull_albero "In cima all'albero"
    with
        description "Ti tieni precariamente appesa al tronco.",
        d_to radura,
    has light;

Object  ramo "ramo largo e robusto" sull_albero
    with
        description "E' abbastanza largo da sostenere un'oggetto.",
        name 'ramo' 'largo' 'robusto',
        each_turn [; if (nido in ramo) deadflag = 2; ],
    has static supporter;

!=============================================================================
! Entry point routines

[ Initialise; location = davanti_baita; ];

!=============================================================================
! Grammatica

Include "ItalianG";

!=============================================================================
```

Codice sorgente del gioco rivisto

```
!% -SD

!=============================================================================
Constant Story "Heidi";
Constant Headline
            "^Semplice esempio in Inform
             ^di Roger Firth e Sonja Kesserich.^";
             ! Traduzione di Paolo Lucchesi
Constant MAX_CARRIED 1;

Include "Parser";
Include "VerbLib";
Include "Replace";

!=============================================================================
! Oggetti

Object  davanti_baita "Di fronte a una baita"
    with
        description
            "Ti trovi davanti a una baita. Verso est si stende la
             foresta.",
        e_to foresta,
        in_to "E' una stupenda giornata, meglio stare all'aperto.",
        cant_go "L'unico sentiero conduce ad est.",
    has light;

Object  baita "piccola baita" davanti_baita
    with
        description "E' piccola e semplice, ma qui tu sei felice.",
        name 'piccola' 'baita' 'case' 'capanna' 'rifugio',
        before [;
            Enter:
                print_ret "E' una stupenda giornata, meglio stare
                           all'aperto.";
        ],
    has scenery female;

Object  foresta "Nel folto del bosco"
    with
        description
            "Attraverso la folta vegetazione, ti pare di scorgere un
             edificio verso ovest. Un sentiero porta verso nord-est.",
        w_to davanti_baita,
        ne_to radura,
    has light;

Object  uccello "uccellino" foresta
    with
        description
            "Troppo giovane per saper volare, il passerotto pigola
             inerme.",
        name 'uccello' 'uccellino' 'passero' 'passerotto' 'volatile',
        before [;
            Listen:
                print "Sembra spaventato e bisognoso d'aiuto.^";
                return true;
        ],
    has ;
```

```
Object  radura "Una radura nella foresta"
    with
        description
             "Un alto sicomoro si erge al centro di questa radura. Il
              sentiero si inoltra tra gli alberi, verso sud-ovest.",
        sw_to foresta,
        u_to sull_albero,
    has light;

Object  nido "nido di uccelli" radura
    with
        description
            "Il nido @`e fatto di rametti e sterpi intrecciati.",
        name 'nido' 'rametti' 'sterpi',
    has container open;

Object  albero "albero di sicomoro" radura
    with
        description
            "Fieramente alto nel mezzo della radura, l'albero sembra
             molto facile da scalare.",
        name 'albero' 'sicomoro' 'tronco',
        before [;
            Climb:
                PlayerTo(sull_albero);
                return true;
        ],
    has scenery;

Object  sull_albero "In cima all'albero"
    with
        description "Ti tieni precariamente appesa al tronco.",
        d_to radura,
        after [;
            Drop:
                move noun to radura;
                return false;
        ],
    has light;

Object  ramo "ramo largo e robusto" sull_albero
    with
        description "E' abbastanza largo da sostenere un'oggetto.",
        name 'ramo' 'largo' 'robusto',
        each_turn [; if (uccello in nido && nido in ramo) deadflag = 2;
        ],
    has static supporter;

!============================================================================
! Entry point routines

[ Initialise; location = davanti_baita; ];

!============================================================================
! Grammatica

Include "ItalianG";

!============================================================================
```

APPENDICE C – La Storia di "GUGLIELMO TELL"

Anche Guglielmo Tell, il nostro secondo gioco, è molto semplice. Vedi "Guglielmo Tell: raccontiamone la storia" nel capitolo sesto, "Guglielmo Tell: i primi anni" nel settimo capitolo, "Guglielmo Tell: il cuore della storia" nel capitolo otto e "Guglielmo Tell: la fine è vicina" nel nono capitolo.

Trascrizione di una partita

```
Il luogo: Altdorf, nel cantone Svizzero di Uri. Siamo nell'anno 1307, e
la Svizzera è sotto il governo dell'imperatore Alberto di Asburgo. Il
governatore locale - il balivo - è il gradasso Hermann Gessler, che ha
posto il suo cappello su di un palo di legno nel centro della piazza
principale; chiuque passi attraverso la piazza deve inchinarsi a questo
odioso simbolo del potere imperiale.

Sei arrivato dalla tua baita sui monti, accompagnato dal tuo figlio più
giovane, per acquistare derrate. Sei un uomo fiero e indipendente, una
guida e un cacciatore, conosciuto sia per la tua abilità come arciere
e, forse poco saggiamente (visto che i suoi soldati sono ovunque), per
essere incapace di nascondere il tuo disprezzo per il balivo.

E' giorno di mercato: la città è piena di gente proveniente dai vicini
villaggi.

Guglielmo Tell
Semplice esempio in Inform
di Roger Firth and Sonja Kesserich.
Versione 3 -- Numero di serie 040804
Inform v6.31 -- Libreria 6/11  --  Infit v2.5    SD

Una strada di Altdorf
La piccola strada conduce verso nord alla piazza principale. La gente
del luogo sta sciamando nella città attraverso la porta a sud,
salutando a voce alta, offrendo prodotti in vendita, scambiando
notizie, informandosi con incredulità esagerata sui prezzi delle merci
esposte dai mercanti, i cui banchi rendono ancora più difficile
l'avanzare in mezzo alla folla.

"Stammi vicino, figliolo," dici, "altrimenti potresti perderti fra
tutta questa gente."

>ESAMINA TUO FIGLIO
Un tranquillo ragazzo biondo di otto primavere, rapido ad imparare lo
stile di vita dei montanari.

>VAI A NORD

Lungo la strada
La gente continua a premere e a farsi strada dalla porta sud alla
piazza principale, che si trova appena più a nord. Riconosci la
proprietaria di un banco di frutta e verdura.

Helga smette di sistemare le patate e ti saluta calorosamente.

"Salve, Guglielmo, è una buona giornata per il commercio! Questo è il
giovane Walter? Come è cresciuto... Ecco, questa è una mela per lui...
Se toglie la parte ammaccata, il resto è buono. Come sta la signora
Tell? Salutamela davvero..."
```

```
>INVENTARIO
Stai portando:
  un mela
  una faretra (indossata)
    tre frecce
  un arco

>PARLA A HELGA
Ringrazi calorosamente Helga per la mela.

[Il tuo punteggio è appena aumentato di un punto.]

>DAI MELA A WALTER
"Grazie, Papi."

[Il tuo punteggio è appena aumentato di un punto.]

>NORD
```

Lato sud della piazza
La piccola strada che conduce verso sud si apre nella piazza principale, per poi riprendere dalla parte opposta di questo affollato luogo di ritrovo. Per continuare lungo la strada, verso la tua destinazione - la conceria di Johansson - devi attraversare, andando verso nord, la piazza, al centro della quale vedi il cappello di Gessler messo su di un palo. Se vuoi andare avanti, non puoi evitare di passarci vicino. Soldati imperiali si fanno largo rudemente tra la folla, spingendo, calciando e imprecando ad alta voce.

```
>ESAMINA SOLDATI
Uomini sgarbati e violenti, non di queste parti.

>ESAMINA IL CAPPELLO
Sei troppo lontano per il momento.

>N
```

In mezzo alla piazza
C'è meno folla al centro della piazza; la maggior parte della gente preferisce tenersi il più lontano possibile dal palo che troneggia in questo luogo, reggendo quell'assurdo cappello cerimoniale. Un gruppo di soldati rimane nei pressi, osservando chiunque passi di qui.

```
>X CAPPELLO
Il palo, il tronco di un piccolo pino, solo pochi centimetri di
diametro, sarà alto tre o quattro metri. Sulla cima è stato appoggiato
con cura il ridicolo cappello di cuoio nero e rosso di Gessler, dalla
larga tesa e adornato con un ciuffo di piume d'oca tinte.

>N
Un soldato ti sbarra la strada.

"Hey, tu, spilungone; hai dimenticato le buone maniere? Forse sarebbe
il caso di fare un bel saluto al cappello del balivo, non trovi?"

>N

"Ti conosco, Tell, sei uno che porta solo guai, vero? Non vogliamo
teste calde qui, quindi fai il bravo ragazzo e porgi il tuo saluto al
dannato cappello. Fallo ora, non voglio chiedertelo di nuovo..."
```

>N

"Va bene, Herr Tell, ora siete nei guai. Ve l'ho chiesto gentilmente, ma voi siete troppo orgoglioso e troppo stupido. Penso che il balivo voglia scambiare qualche parola con voi."

Dicendo ciò, i soldati prendono te e Walter e, mentre il sergente corre via a cercare Gessler, il resto degli uomini vi trascina rudemente verso il vecchio tiglio che cresce nella piazza del mercato.

Piazza del mercato
Il mercato di Altdorf, vicino alla piazza principale, è stato velocemente sgombrato dai banchi. Un gruppo di soldati ha spinto indietro la folla per lasciare spazio libero di fronte al tiglio che è cresciuto qui fin da quando la gente ha memoria. Normalmente esso fa ombra ai vecchi della città che si ritrovano qui per chiacchierare, guardare le ragazze e giocare a carte. Oggi però, esso è solitario... eccetto che per Walter, che è stato legato al tronco. A circa quaranta metri dall'albero, tu sei tenuto bloccato da due degli uomini del balivo.

Gessler sta osservando da lontano, con un sogghigno sul volto.
"A quanto pare hai bisogno di una buona lezione, stupido. Nessuno deve attraversare la piazza senza prestare omaggio a Sua Altezza Imperiale Alberto; nessuno, hai capito? Potrei farti decapitare per tradimento, ma voglio essere clemente. Se ti comporterai ancora follemente, non potrai aspettarti alcuna pietà da parte mia, ma questa volta sei libero di andare... non appena mi avrai dimostrato la tua abilità di arciere colpendo questa mela da dove ti trovi. Non dovrebbe essere troppo difficile. Sergente, prenda; la appoggi sulla testa di quel piccolo bastardo."

>X GESSLER
Basso e robusto, ma dal volto subdolo e affilato, Gessler usa il potere affidatogli opprimendo la comunità locale.

>X WALTER
Ti sta guardando, cercando di apparire coraggioso e rimanere fermo. Le sue braccia sono legate dietro al tronco, e la mela è stata posata tra i suoi capelli biondi.

>X MELA
A questa distanza puoi appena vederla.

>LANCIA UNA FRECCIA ALLA MELA
Con calma e fermezza incocchi una freccia nell'arco, tendi la corda e prendi la mira con più cura di quanto tu abbia mai fatto in vita tua. Trattenendo il fiato, senza un batter d'occhio, timorosamente, rilasci la freccia. Questa vola attraverso la piazza, verso tuo figlio, e pianta la mela contro il tronco dell'albero. La folla esulta; Gessler sembra decisamente deluso.

***** Hai vinto *****

In questa partita hai totalizzato 3 punti su 3 possibili, in 17 turni.

Vuoi RICOMINCIARE, CARICARE una partita salvata o USCIRE ?
>USCIRE

Codice sorgente del gioco

```
!% -SD
!=============================================================================
Constant Story "Guglielmo Tell";
Constant Headline
            "^Semplice esempio in Inform
             ^di Roger Firth and Sonja Kesserich.^";
             ! Traduzione di Paolo Lucchesi
!Release 1; Serial "020428";    ! IBG first edition (public beta)
!Release 2; Serial "020827";    ! IBG second edition
Release 3;  Serial "040804";    ! conto delle release pubbliche

Constant MAX_SCORE = 3;

Include "Parser";
Include "VerbLib";
Include "Replace";

!=============================================================================
! Classi

Class   Room
    has   light;

Class   Prop
    with
        before [;
            Examine: return false;
            default:
                print_ret  "Non  hai  bisogno  di  preoccuparti  ",  (artdi)
                self, ".";
        ],
    has scenery;

Class   Furniture
    with
        before [;
            Take,Pull,Push,PushDir:
                print_ret (The) self, " @`e troppo pesante.";
        ],
    has static supporter;

Class   Arrow
    with
        name 'freccia' 'frecce//p',
        article "una",
        plural "frecce",
        description   "Come tutte le altre tue frecce, dritta e
                       acuminata.",
        before [;
            Drop,Give,ThrowAt:
                print_ret  "Troppo  pericolose,  meglio  non  lasciarle  in
                giro.";
        ];

Class   NPC
    with
        life [;
            Answer,Ask,Order,Tell:
                print_ret "Usa soltanto PARLA [", (arta) self, "].";
```

```
        ],
    has animate;

!=============================================================================
! Oggetti

Room    strada "Una strada di Altdorf"
    with
        description [;
            print  "La piccola strada conduce verso nord alla piazza
                principale. La gente del luogo sta sciamando nella
                citt@`a attraverso la porta a sud, salutando a voce alta,
                offrendo prodotti in vendita, scambiando notizie,
                informandosi con incredulit@`a esagerata sui prezzi delle
                merci esposte dai mercanti, i cui banchi rendono ancora
                pi@`u difficile l'avanzare in mezzo alla folla.^";
        if (self hasnt visited)
            print "^~Stammi vicino, figliolo,~ dici, ~altrimenti
                potresti perderti fra tutta questa gente.~^";
        ],
        n_to vicino_piazza,
        s_to
            "La folla, muovendosi verso la piazza a nord, ti impedisce
             di tornare indietro.";

Prop    "porta meridionale" strada
    with
        name 'cancello' 'porta' 'meridionale',
        description   "Nelle mura della citt@`a si apre una larga
                       porta. Il pesante cancello di legno ora @`e
                       aperto.";

Prop    "banchi"
    with
        name 'banchi',
        description   "Cibo, abiti, attrezzature da montagna; la solita
                       roba.",
        found_in strada vicino_piazza,
    has pluralname;

Prop    "frutti"   !! aggiunta
    with
        name   'beni'   'merci'   'frutti'   'cibo'   'abiti'   'montagna'
               'attrezzature' 'cose' 'roba',
        description "Niente che possa attirare la tua attenzione.",
        found_in strada vicino_piazza,
    has pluralname;

Prop    "mercanti"
    with
        name 'mercante' 'mercanti',
        description
            "Qualche truffatore, ma per il resto gente a posto che si
             guadagna da vivere.",
        found_in strada vicino_piazza,
    has animate pluralname;

Prop    "gente"
    with
        name 'persone' 'gente',
        description "Gente di montagna, come te.",
```

```
        found_in [; return true; ],
    has animate female;

!-----------------------------------------------------------------------

Room    vicino_piazza "Lungo la strada"
    with
        description
            "La gente continua a premere e a farsi strada dalla porta
             sud alla piazza principale, che si trova appena pi@`u a
             nord. Riconosci la proprietaria di un banco di frutta e
             verdura.",
    n_to piazza_sud,
    s_to strada;

Furniture   banco "banco di frutta e verdura" vicino_piazza
    with
        name 'frutta' 'verdura' 'banco' 'tavolo',
        description
            "Davvero un piccolo banco, con un grosso mucchio di patate,
             qualche carota, qualche rapa, un po' di mele.",
        before [; Search: <<Examine self>>; ],
    has scenery;

Prop    "patate" vicino_piazza
    with
        name 'patata' 'patate' 'mucchio',
        description
            "Devono essere state raccolte un po' di tempo fa... almeno
             300 anni!",
    has pluralname female;

Prop    "frutta e verdura" vicino_piazza
    with
        name 'carota' 'carote' 'rapa' 'rape' 'mele' 'vegetali',
        description "Prodotti locali.";

NPC     proprietaria "Helga" vicino_piazza
    with
        name   'proprietaria'   'verduraia'   'venditrice'   'negoziante'
               'mercante' 'helga' 'vestito' 'sciarpa',
        description
            "Helga @`e una donna grassoccia e affabile, infagottata in
             un abito informe e una sciarpa a pois.",
        initial [;
            print "Helga smette di sistemare le patate e ti saluta
                calorosamente.^";
            if (location hasnt visited) {
                move mela to player;
                print "^~Salve, Guglielmo, @`e una buona giornata per il
                    commercio! Questo @`e il giovane Walter? Come @`e
                    cresciuto... Ecco, questa  @`e una
                    mela per lui... Se toglie la parte ammaccata, il
                    resto @`e buono. Come sta la signora Tell?
                    Salutamela davvero...~^";
            }
        ],
        frasi_dette 0,   ! per contare gli argomenti di conversazione
        life [;
            Kiss: print_ret "~Ooh, che sfacciato!~";
            Talk:
```

```
                self.frasi_dette = self.frasi_dette + 1;
                switch (self.frasi_dette) {
                    1:  score = score + 1;
                        print_ret  "Ringrazi calorosamente Helga per la
                                    mela.";
                    2:  print_ret  "~Ci vediamo presto.~";
                    default: return false;
                }
        ],
    has female proper;

!-------------------------------------------------------------------------

Room    piazza_sud "Lato sud della piazza"
    with
        description
             "La piccola strada che conduce verso sud si apre nella
              piazza principale, per poi riprendere dalla parte opposta
              di questo affollato luogo di ritrovo. Per continuare lungo
              la strada, verso la tua destinazione - la conceria di
              Johansson - devi attraversare, andando verso nord, la
              piazza, al centro della quale vedi il cappello di Gessler
              messo su di un palo. Se vuoi andare avanti, non puoi
              evitare di passarci vicino. Soldati imperiali si fanno
              largo rudemente tra la folla, spingendo, calciando e
              imprecando ad alta voce.",
        n_to centro_piazza,
        s_to vicino_piazza;

Prop    "cappello su un palo"        !!cambiato
    with
        name 'cappello' 'palo',
        before [;
            default: print_ret "Sei troppo lontano per il momento.";
        ],
        found_in piazza_sud piazza_nord;

Prop    "soldati di Gessler"
    with
        name 'soldato' 'soldati',
        description "Uomini sgarbati e violenti, non di queste parti.",
        before [;
            FireAt:print_ret  "Sono abbondantemente in sovrannumero.";
            Talk: print_ret   "Una simile feccia non merita la tua
                               considerazione.";
        ],
        found_in piazza_sud piazza_nord centro_piazza mercato,
    has animate pluralname;

!-------------------------------------------------------------------------

Room    centro_piazza "In mezzo alla piazza"
    with
        description
            "C'@`e meno folla al centro della piazza; la maggior parte
             della gente preferisce tenersi il pi@`u lontano possibile
             dal palo che troneggia in questo luogo, reggendo
             quell'assurdo cappello cerimoniale. Un gruppo di soldati
             rimane nei pressi, osservando chiunque passi di qui.",
        n_to piazza_nord,
        s_to piazza_sud,
```

```
        avvertimenti 0,       ! per contare gli avvertimenti dei soldati
        before [;
            Go:
                if (noun == s_obj) {
                    self.avvertimenti = 0;
                    palo.salutato = false;
                }
                if (noun == n_obj) {
                    if (palo.salutato == true) {
                        print "^~Arrivederci, e buona giornata.~^";
                        return false;
                    }   ! fine (palo salutato)
                else {
                    self.avvertimenti = self.avvertimenti + 1;
                    switch (self.avvertimenti) {
                        1:  print_ret "Un soldato ti sbarra la strada. ^^
                                       ~Hey, tu, spilungone; hai
                                       dimenticato le buone maniere?
                                       Forse sarebbe il caso di fare un
                                       bel saluto al cappello del balivo,
                                       non trovi?~";
                        2:  print_ret "^~Ti conosco, Tell, sei uno che
                                       porta solo guai, vero? Non
                                       vogliamo teste calde qui, quindi
                                       fai il bravo ragazzo e porgi il
                                       tuo saluto al dannato cappello.
                                       Fallo ora, non voglio chiedertelo
                                       di nuovo...~";
                        default:
                            print "^~Va bene, ";
                            style underline; print "Herr"; style roman;
                            print " Tell, ora siete nei guai. Ve l'ho
                                   chiesto gentilmente, ma voi siete
                                   troppo orgoglioso e troppo stupido.
                                   Penso che il balivo voglia scambiare
                                   qualche parola con voi.~ ^^ Dicendo
                                   ci@`o, i soldati prendono te e Walter
                                   e, mentre il sergente corre via a
                                   cercare Gessler,
                                   il resto degli uomini vi trascina
                                   rudemente verso il vecchio tiglio che
                                   cresce nella piazza del mercato.^";
                            move mela to figlio;
                            PlayerTo(mercato);
                            return true;
                    }   ! fine switch
                }   ! fine (palo non salutato)
            }   ! fine (noun == n_obj)
        ];

Furniture   palo "cappello su un palo" centro_piazza        !! cambiato
    with
        name   'palo' 'legno' 'tronco' 'pino' 'cappello' 'nero' 'rosso'
               'cuoio' 'tesa' 'piume',
        description
             "Il palo, il tronco di un piccolo pino, solo pochi
              centimetri di diametro, sar@`a alto tre o quattro metri.
              Sulla cima @`e stato appoggiato con cura il ridicolo
              cappello di cuoio nero e rosso di Gessler, dalla larga
              tesa e adornato con un ciuffo di piume d'oca tinte.",
        salutato false,
```

```
        before [;
            FireAt:   !! aggiunto
                print_ret "Allettante, ma non sei in cerca di guai.";
            Salute:
                self.salutato = true;
                print_ret "Saluti il cappello sull'alto palo. ^^ ~Grazie
                          davvero, messere~, sghignazzano i soldati.";
        ],
    has scenery;

!-----------------------------------------------------------------------

Room    piazza_nord "Lato nord della piazza"
    with
        description
            "Una piccola strada conduce verso nord, lasciando la piazza
             affollata. Al centro della piazza, di poco pi@`u a sud,
             vedi ancora il palo e il cappello.",
        n_to [;
            deadflag = 3;
            print_ret  "Con Walter al tuo fianco, lasci la piazza
                        percorrendo la strada verso nord, andando verso
                        la conceria di Johansson.";
        ],
        s_to "Non vuoi assolutamente passarci di nuovo.";

!-----------------------------------------------------------------------

Room    mercato "Piazza del mercato"
    with
        description
            "Il mercato di Altdorf, vicino alla piazza principale, @`e
             stato velocemente sgombrato dai banchi. Un gruppo di
             soldati ha spinto indietro la folla per lasciare spazio
             libero di fronte al tiglio che @`e cresciuto qui fin da
             quando la gente ha memoria. Normalmente esso fa ombra ai
             vecchi della citt@`a che si ritrovano qui per
             chiacchierare,guardare le ragazze e giocare a carte. Oggi
             per@`o, esso @`e solitario... eccetto che per Walter, che
             @`e stato legato al tronco. A circa quaranta metri
             dall'albero, tu sei tenuto bloccato da due degli uomini del
             balivo.",
        cant_go "Cosa? E lasceresti tuo figlio legato a quell'albero?";

Object  albero "tiglio" mercato
    with
        name 'albero' 'tiglio' 'tronco',
        description "Un grosso albero.",
        before [;
            FireAt:
                if (BowOrArrow(second) == true) {
                    deadflag = 3;
                    print_ret "La mano ti trema, e la freccia vola alta,
                               andando a colpire il tronco, almeno una
                               spanna sopra la testa di Walter.";
                }
                return true;
        ],
    has scenery;
```

```
NPC     balivo "balivo" mercato
    with
        name 'governatore' 'balivo' 'hermann' 'gessler',
        description
            "Basso e robusto, ma dal volto subdolo e affilato, Gessler
             usa il potere affidatogli opprimendo la comunit@`a
             locale.",
        initial [;
            print  "Gessler sta osservando da lontano, con un sogghigno
                    sul volto.^";
            if (location hasnt visited)
            print  "~A quanto pare hai bisogno di una buona lezione,
                    stupido. Nessuno deve attraversare la piazza senza
                    prestare omaggio a Sua Altezza Imperiale Alberto;
                    nessuno, hai capito? Potrei farti decapitare per
                    tradimento, ma voglio essere clemente. Se ti
                    comporterai ancora follemente, non potrai aspettarti
                    alcuna piet@`a da parte mia, ma questa volta sei
                    libero di andare... non appena mi avrai dimostrato
                    la tua abilit@`a di arciere colpendo questa mela da
                    dove ti trovi. Non dovrebbe essere troppo difficile.
                    Sergente, prenda; la appoggi sulla testa di quel
                    piccolo bastardo.~^";
        ],
        life [;
            Talk:  print_ret  "Non hai intenzione di rivolgergli la
                               parola.";
        ],
        before [;
            FireAt:
            if (BowOrArrow(second) == true) {
                deadflag = 3;
                print_ret "Prima che i soldati possano reagire, ti volti
                           e scagli una freccia contro Gessler; la tua
                           frecca colpisce il suo cuore ed egli muore
                           miseramente.  La  folla  ha  un    sussulto,
                           seguito da un applauso.";
            }
            return true;
        ],
    has male;

!=====================================================================
! Oggetti del giocatore

Object  arco "arco"
    with
        name 'arco',
        description "Il tuo fidato arco di tasso.",
        before [;
            Drop,Give,ThrowAt:
                print_ret "Non ti separi mai dal tuo fidato arco.";
        ]
    has clothing;

Object  faretra "faretra"
    with
        name 'faretra',
        description
            "Fatta in pelle di capra, di solito pende dalla tua spalla
             sinistra.",
```

```
        before [;
            Drop,Give,ThrowAt:
                print_ret "Ma @`e un regalo di tua moglie Hedwig.";
        ],
    has container open clothing female;

Arrow   "freccia" faretra;
Arrow   "freccia" faretra;
Arrow   "freccia" faretra;

NPC     figlio "tuo figlio"
    with
        name 'figlio' 'tuo' 'ragazzo' 'bambino' 'walter',
        description [;
            if (location == mercato)
                print_ret  "Ti sta guardando, cercando di apparire
                           coraggioso e rimanere fermo. Le sue braccia
                           sono legate dietro al tronco, e la mela @`e
                           stata posata tra i suoi capelli biondi.";
            else
                print_ret  "Un tranquillo ragazzo biondo di otto
                            primavere, rapido ad imparare lo stile di
                            vita dei montanari.";
        ],
        life [;
            Give:
                score = score + 1;
                move noun to self;
                print_ret "~Grazie, Papi.~";
            Talk:
                if (location == mercato)
                    print_ret  "~Stai calmo, figliolo, e confida in
                                Dio.~";
                else
                    print_ret  "Spieghi a tuo figlio la tua visione della
                                vita.";
        ],
        before [;
            Examine,Listen,Salute,Talk: return false;
            FireAt:
                if (location == mercato) {
                    if (BowOrArrow(second)) {
                        deadflag = 3;
                        print_ret  "Oops! Sicuramente non volevi far
                                    ci@`o...";
                    }
                    return true;
                }
                else return false;
            default:
                if (location == mercato)
                    print_ret "Le guardie non lo permetterebbero.";
                else return false;
        ],
        found_in [; return true; ],
    has male proper scenery transparent;

Object  mela "mela"
    with
        name 'mela',
        description [;
```

```
            if (location == mercato)
                print_ret  "A questa distanza puoi appena vederla.";
            else
                print_ret  "La mela @`e a chiazze verdi e marroni.";
        ],
        before [;
            Drop:
                print_ret  "Una   mela   vale   sempre   qualcosa,   meglio
                           tenersela.";
                Eat:   print_ret "Helga te l'ha data per Walter...";
                FireAt:
                    if (location == mercato) {
                        if (BowOrArrow(second)) {
                            score = score + 1;
                            deadflag = 2;
                            print_ret  "Con calma e fermezza incocchi una
                                       freccia nell'arco, tendi la corda
                                       e prendi la mira con pi@`u cura di
                                       quanto tu abbia mai fatto in vita
                                       tua. Trattenendo il fiato, senza
                                       un batter d'occhio, timorosamente,
                                       rilasci    la    freccia.    Questa
                                       attraverso la piazza, verso tuo
                                       figlio,e la mela contro il tronco
                                       dell'albero.  La  esulta;  Gessler
                                       sembra decisamente deluso.";
                        }
                        return true;
                    }
                    else return false;
        ];
!=====================================================================
! Routine Entry point

[ Initialise;
    location = strada;
    lookmode = 2;       ! in modo VERBOSO
    move arco to player;
    move faretra to player; give faretra worn;
    player.description =
         "Indossi i tradizionali abiti dei montanari svizzeri.";
    print_ret  "^^ Il luogo: Altdorf, nel cantone Svizzero di Uri. Siamo
                nell'anno 1307, e la Svizzera @`e sotto il governo
                dell'imperatore Alberto di Asburgo. Il governatore
                locale - il balivo - @`e il gradasso Hermann Gessler,
                che ha posto il suo cappello su di un palo di legno nel
                centro della piazza principale; chiuque passi attraverso
                la piazza deve inchinarsi a questo odioso simbolo del
                potere imperiale. ^^ Sei arrivato dalla tua baita sui
                monti, accompagnato dal tuo figlio pi@`u giovane, per
                acquistare derrate. Sei un uomo fiero e indipendente,
                una guida e un cacciatore, conosciuto sia per la tua
                abilit@`a come arciere e, forse poco saggiamente (visto
                che i suoi soldati sono ovunque), per essere incapace di
                nascondere il tuo disprezzo per il balivo. ^^ E' giorno
                di mercato: la citt@`a @`e piena di gente proveniente
                dai vicini villaggi.^";
];

[ DeathMessage; print "Hai rovinato una bellissima leggenda."; ];
```

```
!=============================================================================
! Grammatica standard e estensioni

Include "ItalianG";

!-----------------------------------------------------------------------------

[ TalkSub;
    if (noun == player) print_ret "Niente di quello che dici pu@`o
                                   sorprenderti.";
    if (RunLife(noun,##Talk) ~= false) return;  ! consulta life[;Talk:]
    print_ret "Al momento, non ti viene niente da dire.";
];


Verb 'chiacchiera' 'conversa' 'c//'
    *   'con' creature                  -> Talk;

Extend 'parla' first
    *   'a'/'ad'/'all^'/'allo'/'alla'/'al'/
        'agli'/'ai'/'alle'/'con' creature
                                        -> Talk;

!-----------------------------------------------------------------------------

[ BowOrArrow o;
    if (o == arco or nothing || o ofclass Arrow) return true;
    print "Non @`e granch@`e come arma, vero?";
    return false;
];

[ FireAtSub;
    if (noun == nothing)
        print_ret "Cosa? Scagliare frecce a caso?";
    if (BowOrArrow(second) == true)
        print_ret "Sembra pericoloso, non trovi?";
];

Verb 'spara' 'mira' 'scaglia'
    *                                   -> FireAt
    *   noun                            -> FireAt
    *   'con' noun                      -> FireAt
    *   'a'/'ad'/'all^'/'allo'/'alla'/'al'/'agli'/'ai'/
       'alle'/'su'/'sul'/'sullo'/'sull^'/'sulla'/'sui'/
       'sugli'/'sulle'/'sopra'/'contro' noun
                                        -> FireAt
    *   'a'/'ad'/'all^'/'allo'/'alla'/'al'/'agli'/'ai'/
        'alle'/'su'/'sul'/'sullo'/'sull^'/'sulla'/'sui'/
        'sugli'/'sulle'/'sopra'/'contro' noun
        'con' noun
                                        -> FireAt
    *   'con' noun 'a'/'ad'/'all^'/'allo'/'alla'/'al'/'agli'/
        'ai'/'alle'/'su'/'sul'/'sullo'/'sull^'/'sulla'/
        'sui'/'sugli'/'sulle'/'sopra'/'contro' noun
                                        -> FireAt reverse
    *   noun 'a'/'ad'/'all^'/'allo'/'alla'/'al'/'agli'/'ai'/
        'alle'/'su'/'sul'/'sullo'/'sull^'/'sulla'/'sui'/
        'sugli'/'sulle'/'sopra'/'contro' noun
                                    -> FireAt reverse;

Extend only 'lancia' first
```

```
    * noun 'a'/'ad'/'all^'/'allo'/'alla'/'al'/'agli'/'ai'/
      'alle'/'su'/'sul'/'sullo'/'sull^'/'sulla'/'sui'/
      'sugli'/'sulle'/'sopra'/'contro' noun
                                  -> FireAt reverse;

!---------------------------------------------------------------------
[ SaluteSub;
   if (noun has animate)
      print_ret (The) noun, " restituisce il saluto.";
   print_ret (The) noun, " non mostra alcuna reazione.";
];

Verb 'inchinati' 'genuflettiti'
    *   'a'/'ad'/'all^'/'allo'/'alla'/'al'/'agli'/'ai'/
        'alle'/'verso' noun
                                      -> Salute;

Verb 'riverisci'
    *   noun                          -> Salute;

Extend 'saluta' first
    *   noun                          -> Salute;

Extend 'dai'
    *   'omaggio'/'omaggi' 'a'/'ad'/'all^'/'allo'/
        'alla'/'al'/'agli'/'ai'/'alle'/'verso' noun
                                      -> Salute;

!---------------------------------------------------------------------
[ UntieSub; print_ret "Non dovresti provarci."; ];

Verb 'slega' 'sciogli' 'libera' 'rilascia'
    *   noun                          -> Untie;

!=====================================================================
```

Compilare strada facendo

Il vostro apprendimento di come funziona il gioco di "Guglielmo Tell" sarà notevolmente facilitato se durante la lettura di questa guida dedicherete del tempo a digitare il codice sul vostro computer. A noi ci sono voluti ben quattro capitoli per descrivere il gioco, che pertanto rimane incompleto e non giocabile fino alla fine del Capitolo 9. Perciò, anche se non fate errori di digitazione, il sorgente non potrà essere compilato senza errori se non si arriva a quel punto, a causa dei riferimenti nei precedenti paragrafi a oggetti che non sono presenti se non nei capitoli successivi (ad esempio il Capitolo 6 menziona l'arco e le frecce, ma essi non vengono definiti fino al capitolo 7). Ciò rappresenta una piccola seccatura, visto che vi abbiamo consigliato generalmente di compilare frequentemente gli esempi – più o meno dopo ogni cambiamento che apportate al gioco – così da poter rilevare e correggere gli errori di sintassi il più facilmente possibile.

Fortunatamente, esiste un sistema piuttosto semplice di aggirare questa difficoltà anche se comporta quache piccolo imbroglio. Il trucco è aggiungere temporaneamente delle definizioni minime – spesso chiamate "mozziconi" (stabs) – di oggetti alla cui definizione completa si provvederà solo in seguito.

Per esempio, se provate a compilare il gioco nello stato in cui si trova alla fine del sesto capitolo, otterrete questo:

```
Tell.inf(16): Warning: Class "Room" declared but not used
Tell.inf(19): Warning: Class "Prop" declared but not used
Tell.inf(27): Warning: Class "Furniture" declared but not used
Tell.inf(44): Error: No such constant as "street"
Tell.inf(46): Error: No such constant as "bow"
Tell.inf(47): Error: No such constant as "quiver"
Compiled with 3 errors and 3 warnings
```

Comunque, aggiungendo queste linee alla fine del file sorgente:

```
[TYPE]
    ! ================================================================
    ! DEFINIZIONI TEMPORANEE NECESSARIE ALLA COMPILAZIONE ALLA FINE DEL
    ! CAPITOLO 6


    Room strada;
    Object arco;
    Object faretra;
```

una nuova compilazione ora dovrebbe riportare solo questo:

```
Tell.inf(19): Warning: Class "Prop" declared but not used
Tell.inf(27): Warning: Class "Furniture" declared but not used
Compiled with 2 warnings
```

Che è molto meglio. Non bisogna preoccuparsi per questi avvertimenti, dal momento che è facile capire da dove provengono; e ad ogni modo essi saranno eliminati tra poco. La cosa più importante è che non vi siano errori, il che significa che probabilmente non avete fatto nessuno dei più evidenti errori di sintassi o di digitazione. Significa anche che il compilatore ha creato un file `Z5`, così ora potete provare a "giocare" una partita. Se ci provate otterrete questo:

```
Guglielmo Tell
Semplice esempio in Inform
di Roger Firth e Sonja Kesserich.
Versione 3 / Numero di Serie 040804 / Inform v6.30 Libreria 6/11
Infit Versione 2.5 / SD

(strada)
** Library error 11 (27,0) **
** The room "(strada)" has no "description" property **
>
```

ooops! Abbiamo infranto una delle regole di Inform che dice che ogni locazione deve avere una proprietà `description`, che l'interprete visualizzerà quando si entra in quella stanza. Il nostro mozzicone per la locazione strada non possiede una descrizione, così sebbene l'avventura venga compilata con successo, verrà comunque riportato un errore durante l'esecuzione del gioco. Il modo migliore di superare questo ostacolo è di estendere la definizione della nostra classe Room, così:

```
[TYPE]
    Class Room
       with description "LAVORI IN CORSO",
       has light;
```

Facendo in questo modo, ci assicuriamo che ogni locazione abbia una descrizione di qualche genere; normalmente andremmo a sovrascrivere questo valore di partenza con qualcosa di significativo – "La piccola strada conduce verso nord alla piazza principale..." e così via – includendo una proprietà `description` nella definizione di un oggetto. Comunque, in un oggetto mozzicone usato solo per il test dell'avventura, una descrizione di default è sufficiente (e dà meno problemi):

```
    Guglielmo Tell
    Semplice esempio in Inform
    di Roger Firth e Sonja Kesserich.
    Versione 3 / Numero di Serie 040804 / Inform v6.30 Libreria 6/11
    Infit Versione 2.5 / SD


    (strada)
    LAVORI IN CORSO

    >INVENTARIO
    Stai portando:
        una (faretra) (indossata)
        un (arco)

    >ESAMINA FARETRA
    Non puoi vedere niente del genere.

    >
```

Noterete che ci sono un paio di punti interessanti. Poiché non abbiamo dato un nome esterno alla nostra strada, all'arco e alla faretra nei nostri mozziconi, il compilatore ha provveduto per noi ad aggiungere – (strada), (arco) e (faretra) – il nome interno che abbiamo usato nel codice ponendolo tra parentesi. Inoltre, visto che i nostri mozziconi di arco e faretra non hanno una proprietà `name`, non possiamo al momento riferirci a questi oggetti durante la partita. Nessuno di questi problemi è accettabile in un gioco completo, ma al fine di testare l'avventura nei suoi primi passi essi sono facilmente tollerabili.

Precedentemente abbiamo visto come l'aggiunta della definizione temporanea di tre oggetti ci permetta di compilare un gioco incompleto, nello stato in cui è alla fine del Capitolo 6. Ma una volta che abbiamo raggiunto la fine del Capitolo 7, le cose sono piuttosto cambiate, e ora abbiamo bisogno di un set differente di mozziconi.

Per una verifica di compilazione a questo punto, rimuovete il precedente set di mozziconi, e invece aggiungete questi – l'oggetto `piazza_sud` e la `mela`, e una gestione stupida per gestire l'azione `Talk` nella proprietà `life` di Helga:

```
[TYPE]
    ! ================================================================
    ! DEFINIZIONI TEMPORANEE NECESSARIE ALLA COMPILAZIONE ALLA FINE DEL
    ! CAPITOLO 7

    Room piazza_sud;
    Object mela;
    [ TalkSub; ];
```

Allo stesso modo, al termine del Capitolo 8, rimpiazzate i precedenti mozziconi con questi se intendete provare a compilare il gioco:

```
[TYPE]
    ! ================================================================
    ! DEFINIZIONI TEMPORANEE NECESSARIE ALLA COMPILAZIONE ALLA FINE DEL
    ! CAPITOLO 8

    Room mercato;
    Object mela;
    NPC figlio;

    [ TalkSub; ];
    [ FireAtSub; ];
    [ SaluteSub; ];
```

Finalmente, al termine del Capitolo 9 il gioco sarà completo e potrete cancellare completamente i mozziconi.

Usata con cautela, questa tecnica di creare un minimo di mozziconi può essere conveniente: essa permette di "provare" una porzione del gioco attraverso l'interprete, e di compilare e testare il gioco in quello stato, senza aver bisogno di creare tutte le definizioni degli oggetti.

Naturalmente non potrete avere un granché d'interazione con gli oggetti mozziconi, pertanto cercate di non crearne molti. E soprattutto non dimenticatevi di eliminarli ogni volta che avete completato tutte le definizioni.

APPENDICE D – La Storia di "CAPITAN FATO"

Capitan Fato è il nostro terzo ed ultimo gioco ed è un po' più lungo e complesso dei suoi predecessori. Vedi "Capitan Fato: prima!" nel Capitolo 10, "Capitan Fato: seconda!" nel Capitolo 11, "Capitan Fato: terza" nel Capitolo 12 e "Capitan Fato: scena finale!" nel Capitolo 13.

Trascrizione di una partita

Impersonando il tranquillo John Covarth, assistente garzone in una insignificante drogheria, ti FERMI di colpo quando il tuo udito finissimo decifra una chiamata radio della POLIZIA. Un FOLLE sta attaccando la popolazione al Parco Granaio! Devi indossare velocemente il tuo costume da Capitan FATO...!

```
Capitan FATO
Semplice esempio in Inform
di Roger Firth and Sonja Kesserich.
Versione 3 -- Numero di serie 040804
Inform v6.31 -- Libreria 6/11  --  Infit v2.5    SD

In strada
Da una parte, che grazie al tuo SOVRUMANO senso della direzione sai
essere il NORD, c'è un bar in cui si può anche pranzare a quest'ora.
Verso sud, vedi una cabina del telefono.

>ESAMINA ME
Negli abiti della tua identità segreta, riesci in maniera estremamente
efficace a sembrare un perdente, un perfetto imbranato.

>INVENTARIO
Stai portando:
  i tuoi vestiti (indossati)
  il tuo costume

>X COSTUME
Manifattura allo STATO DELL'ARTE, 100% COTONELASTICO(tm) rinforzato
chimicamente.

>TOGLI I VESTITI
In mezzo alla strada? Questo sarebbe uno SCANDALO, e inoltre
rivelerebbe la tua identità segreta.

>X CABINA
Il vecchio pittoresco modello giallo, con spazio per una sola persona.

>ENTRA NELLA CABINA
Con velocità implausibile, ti fiondi all'interno della cabina.

>TOGLI I VESTITI
Non avendo la super-velocità di Superman, realizzi che sarebbe
sconveniente cambiarti sotto gli occhi delle persone che passano.

>ESCI
```

Sei uscito dalla cabina del telefono.

In strada
Da una parte, che grazie al tuo SOVRUMANO senso della direzione sai essere il NORD, c'è un bar in cui si può anche pranzare a quest'ora. Verso sud, vedi una cabina del telefono.

>X BAR
Il miglior bar della città per uno spuntino veloce. Il bar di Benny ha un look da nave spaziale anni '50

>ENTRA NEL BAR
Con un impressionante commistione di fretta e nonchalance entri all'interno del bar.

Il bar di Benny
Benny offre la MIGLIORE selezione di pezzi dolci e sandwich. I clienti riempiono il bancone dove Benny in persona riesce a servire, cucinare e riscuotere senza la minima esistazione. Sulla parete nord del bar vedi una porta rossa che conduce al gabinetto.

>APRI LA PORTA
Sembra essere chiusa a chiave.

>GUARDALA
Una porta rossa con le inequivocabili silhouette di un uomo e di una donna che segnano l'ingresso ai locali igienici. C'è una nota scarabocchiata attaccata alla superficie della porta.

>LEGGI LA NOTA
Rivolgi la tua visione a ULTRAFREQUENZA AVANZATA verso la nota e socchiudi gli occhi concentrandoti, arrendendoti solo quando i bordi della nota iniziano ad annerirsi sotto l'incredibile intensità del tuo sguardo infuocato. Rifletti ancora una volta su quanto sarebbe stato utile se tu avessi mai imparato a leggere.

Una vecchia signora premurosa si avvicina e ti spiega: "Devi chiedere la chiave a Benny, al bancone."

Ti volti verso di lei e inizi a dire: "Oh, la SAPEVO, ma..."

"Di nulla, figliolo," dice la signora mentre esce dal bar.

>X BENNY
Un uomo ingannevolmente GRASSO dotato di incredibile agilità, Benny intrattiene i clienti schiacciandosi noci di cocco sulla fronte quando è dell'umore giusto.

>CHIEDI A BENNY LA CHIAVE
"Il gabinetto è solo per i clienti." mormora, indicando con il dito il menu alle sue spalle.

>X MENU
Il menu appeso al muro elenca tutti i cibi e bevande che Benny può servire. Peccato che tu non abbia mai imparato a leggere, ma fortunatamente c'è il disegno di una grossa tazza di caffè fra tutte le altre scritte incomprensibili.

>BENNY, DAMMI UN CAFFE
In sole due mosse aggraziate, Benny posa di fronte a te il suo famosissimo Caffè macchiato.

>BENNY, DAMMI LA CHIAVE
Benny getta la chiave delle toilettes sul bancone, da cui tu la prendi con un destro e preciso movimento della tua mano SUPER-AGILE.

>APRI LA PORTA CON LA CHIAVE
Apri la porta che conduce al gabinetto.

>N

Un gabinetto unisex
Una stanza quadrata, incredibilmente PULITA, dalle pareti ricoperte di mattonelle di ceramica, che non contiene molto di più di un gabinetto e un interruttore. L'unica uscita è a sud, attraverso la porta che riconduce al bar.

[Il tuo punteggio è appena aumentato di un punto.]

>PREMI L'INTERRUTTORE
Accendi la luce del gabinetto.

>CHIUDI PORTA
Hai chiuso la porta che conduce al bar.

>CHIUDI LA PORTA A CHIAVE
Ora hai chiuso a chiave la porta che conduce al bar.

>X GABINETTO
L'ultima persona ha CIVILMENTE tirato l'acqua dopo aver usato il gabinetto, ma ha dimenticato di raccogliere la PREZIOSA moneta che è caduta dai suoi pantaloni.

>PRENDI MONETA
Ti accosci nella posizione del DRAGO DORMIENTE e senza indugio raccogli e intaschi la moneta con la PRESA SOVRANA.

[Il tuo punteggio è appena aumentato di un punto.]

>TOGLI I VESTITI
Ti togli rapidamente i vestiti e li raccogni in un pacco ULTRA-MINUSCOLO facilmente trasportabile. Poi spieghi il tuo costume in COTONE INVULNERABILE e ti trasformi in Capitan FATO, difensore della libertà e avversario della tirannia!

>APRI LA PORTA CON LA CHIAVE
Apri la porta che conduce al bar.

>S

Il bar di Benny
Benny offre la MIGLIORE selezione di pezzi dolci e sandwich. I clienti riempiono il bancone dove Benny in persona riesce a servire, cucinare e riscuotere senza la minima esistazione. Sulla parete nord del bar vedi una porta rossa che conduce al gabinetto.

I clienti osservano il tuo costume con evidente curiostità.

Sopra il bancone vedi una tazza di caffè.

"Non sapevo ci fosse il circo in città," dice un clienti ad un altro. "Sembra che i pagliacci abbiano il giorno libero."

>BEVI IL CAFFE
Prendi la tazzina e ne bevi un sorso. La REPUTAZIONE MONDIALE di Benny è ben meritata. Appena finisci, Benny porta via la tazzina. "Il caffè viene un'euro, signore."

>DAI LA MONETA A BENNY
Con meravigliose movenze da ILLUSIONISTA, fai apparire una moneta dal tuo costume come se fosse uscita fuori dall'orecchio di Benny! Le persone attorno a te applaudono educatamente. Benny prende la moneta, e la morde SOSPETTOSO. "Grazie, signore. Torni quando vuole," dice.

>DAI LA CHIAVE A BENNY
Benny annuisce mentre tu MIRABILMENTE gli rendi la chiave.

"Questi stilisti non sanno più che fare per farsi conoscere," sbuffa un signore corpulento guardando nella tua direzione. Quelli che hanno sentito cercano di nascondere il sogghigno.

>S
Esci in strada, dove le persone di passaggio riconoscono la STRAVAGANZA arcobaleno del costume di Capitan FATO e gridano il tuo nome con stupore mentre tu SALTI con forza sensazionale nel cielo BLU del mattino!

***** Voli via, diretto a RISOLVERE la SITUAZIONE! *****

In questa partita hai totalizzato 2 punti su 2 possibili, in 33 turni.

Vuoi RICOMINCIARE, CARICARE una partita salvata o USCIRE ?
>USCIRE

Il codice sorgente del gioco

```
!% -SD
!============================================================================
Constant Story "Capitan FATO";
Constant Headline
            "^Semplice esempio in Inform
             ^di Roger Firth and Sonja Kesserich.^";
             ! Traduzione di Paolo Lucchesi
!Release 1; Serial "020428";    ! prima edizione IBG (public beta)
!Release 2;  Serial "020827";   ! seconda edizione IBG
Release 3;  Serial "040804";    ! conto delle release pubbliche

Constant MANUAL_PRONOUNS;
Constant MAX_SCORE     2;
Constant OBJECT_SCORE  1;
Constant ROOM_SCORE    1;

Replace MakeMatch;              ! richiesto da pname.h
Replace Identical;
Replace NounDomain;
Replace TryGivenObject;

Include "Parser";
Include "pname";                ! pname.h la trovate nell'Archivio

Object  LibraryMessages  ! devono essere definiti tra  Parser e VerbLib
    with
        before [;
            Buy:   "Il piccolo commercio ti ha raramente interessato.";
            Dig:   "I tuoi super-sensi non percepiscono NIENTE sotto
                    terra che possa interessarti in questo momento.";
            Pray:  "Non hai bisogno di disturbare DIVINITA' onnipotenti
                   per risolvere la situazione.";
            Sing:  "Ahim@`e! Questo non @`e uno dei tuoi superpoteri.";
            Sleep: "Un eroe @`e SEMPRE all'erta.";
            Sorry: "Capitan FATO non ha tempo per le scuse, ma solo
                   perl'AZIONE.";
            Strong:"Un vocabolario non adatto ad un EROE come te.";
            Swim:  "Rivolgi la tuta ATTENZIONE alla ricerca di un posto
                    adatto per ESERCITARE il tuo superiore stile di
                    nuoto ma, aihm@`e, non trovi niente di simile.";
            Miscellany:
                if (lm_n == 19)
                    if (vestiti has worn)
                        "Negli abiti della tua identit@`a segreta, riesci
                         in maniera estremamente efficace a sembrare un
                         perdente, un perfetto imbranato.";
                    else
                        "Ora che indossi il tuo costume, proietti
                         l'immagine di PURA potenza , di MUSCOLI gonfi e
                         multicolorati, e di uno stile ARDITO e SOBRIO
                         allo stesso tempo.";
                if (lm_n == 38)
                    "Non hai bisogno di questo verbo per risolvere con
                     SUCCESSO la situazione.";
                if (lm_n == 39)
                    "Non @`e qualcosa di cui tu abbia bisogno per
                     RISOLVERE la situazione.";
        ];
```

```
Include "Verblib";
Include "replace";

!=============================================================================
! Classi

Class   Room
    with description "IN COSTRUZIONE",
    has light;

Class   Appliance
    with
        before [;
            Take,Pull,Push,PushDir:
                "Anche se i tuoi muscoli SCOLPITI e adamantini ne
                 sarebbero in grado, tu sei contrario ai danni alla
                 altrui propriet@`a.";
        ],
    has scenery;

!=============================================================================
! Oggetti

Room    strada "In strada"
    with
        name   'citt@`a' 'citta' 'citta^' 'edifici' 'palazzi' 'negozi'
               'appartamenti' 'macchine' 'automobili',
        description [;
            if (player in cabina)
                "Da questo punto STRATEGICO ottieni una visuale completa
                 di tutto il marciapiede e dell'ingresso al bar di
                 Benny.";
            else
                "Da una parte, che grazie al tuo SOVRUMANO senso della
                 direzione sai essere il NORD, c'@`e un bar in cui si
                 pu@`o anche pranzare a quest'ora. Verso sud, vedi una
                 cabina del telefono.";
        ],
        before [;
            Go: if (player in cabina && noun == n_obj) <<Exit cabina>>;
        ],
        n_to [; <<Enter fuori_dal_bar>>; ],
        s_to [; <<Enter cabina>>; ],
        in_to "Va bene, ma da che parte?",
        cant_go
            "Non c'@`e tempo per esplorare. Ti muoverai molto pi@`u
             velocemente nel tuo costume da Capitan FATO."
    has female;

Object  "pedoni" strada
    with
        name 'gente' 'persone' 'pedoni',
        description "Soltanto GENTE che bada ai propri ONESTI affari.",
        before [;
            Examine: return false;
            default: "La gente non sembra considerarti minimamente.";
        ],
    has animate pluralname scenery;
```

```
Appliance cabina "cabina del telefono" strada
    with
        name   'vecchia'    'gialla'    'pittoresca'    'cabina'    'del'
               'telefono',
        description
            "Il vecchio pittoresco modello giallo, con spazio per una
             sola persona.",
        before [;
            Open:  "La cabina @`e gi@`a aperta.";
            Close: "Non c'@`e modo di chiudere la cabina.";
        ],
        after [;
            Enter:
                "Con velocit@`a implausibile, ti fiondi all'interno della
                 cabina.";
        ],
    has enterable container open female;

Appliance "marciapiede" strada
    with
        name 'marciapiede' 'selciato' 'strada',
        article "il",
        description
            "Esegui un veloce controllo del marciapiede e scopri, con
             tua immensa sorpresa, che @`e in TUTTO simile ad ogni altro
             marciapiede della CITTA'!";

Appliance fuori_dal_bar "Il bar di Benny" strada
    with
        name 'bar' 'di' 'benny' 'locale' 'entrata',
        description
            "Il miglior bar della citt@`a per uno spuntino veloce. Il
             bar di Benny ha un look da nave spaziale anni '50",
        before [;
            Enter:
                print  "Con un impressionante commistione di fretta e
                        nonchalance entri all'interno del bar.^";
                PlayerTo(bar);
                return true;
        ],
    has enterable proper;

!-----------------------------------------------------------------------

Room    bar "Il bar di Benny"
    with
        description
            "Benny offre la MIGLIORE selezione di pezzi dolci e
             sandwich. I clienti riempiono il bancone dove Benny in
             persona riesce a servire, cucinare e riscuotere senza la
             minima esistazione. Sulla parete nord del bar vedi una
             porta rossa che conduce al gabinetto.",
        before [;
            Go:
                if (noun ~= s_obj) return false;
                if  (benny.caffe_non_pagato == true ||
                    benny.chiave_non_resa == true) {
                        print  "Come accenni ad uscire per strada, la
                                grossa mano di Benny si appoggia sulla
                                tua spalla.";
```

```
                    if  (benny.caffe_non_pagato == true &&
                        benny.chiave_non_resa == true)
                            "^^~Hey! Hai ancora la mia chiave e non
                             hai pagato il caff@`e. Ti sembro forse
                             uno stupido?~ Ti scusi come
                             soltanto un EROE sa fare e torni
                             all'interno.";
                    if (benny.caffe_non_pagato == true)
                       "^^~Aspetta un minuto, Amico,~ dice.
                        ~Stiamo cercando di sgattaiolare via senza
                        pagare, vero?~ Mormori velocemente una scusa e
                        torni all'interno del bar. Benny torna ai suoi
                        compiti continuando a guardarti sospettoso.";
                    if (benny.chiave_non_resa == true)
                       "^^~Dove credi di andare con la chiave del
                        gabinetto?~ dice. ~Sei forse un ladro?~ Mentre
                        Benny ti spinge di nuovo all'interno del
                        locale, lo rassicuri velocemente spiegando che
                        si @`e trattato solo di uno STUPEFACENTE
                        errore.";
                }
                if (costume has worn) {
                    deadflag = 5;               ! hai vinto!
                      "Esci in strada, dove le persone di passaggio
                       riconoscono la STRAVAGANZA arcobaleno del costume
                       di Capitan FATO e gridano il tuo nome con stupore
                       mentre tu SALTI con forza sensazionale nel cielo
                       BLU del mattino!";
                }
        ],
        appenauscito false,  ! Prima apparizione di Capitan Fato?
        after [;
            Go: !il giocatore è appena arrivato, viene dal bagno?
                if (noun ~= s_obj) return false;
                if (costume has worn && self.appenauscito == false) {
                    self.appenauscito = true;
                    StartDaemon(clienti);
                }
        ],
        s_to strada,
        n_to porta_del_gabinetto;

Appliance bancone "bancone" bar
    with
        name 'bancone' 'banco',
        article "il",
        description
            "Il bancone @`e fatto con una strabiliante LEGA di metalli,
             a PROVA di briciole e liquidi VERSATI e FACILE da pulire. I
             clienti si godono i loro spuntini con ESTREMA
             tranquillit@`a, sicuri sapendo che il bancone pu@`o
             resistere a tutto.",
        before [; Receive: <<Give noun Benny>>; ],
    has supporter;

Object  cibo "Gli spuntini di Benny" bar
    with
        name 'pezzi' 'dolci' 'cibo' 'sandwich' 'pezzo' 'dolce' 'pasta'
             'paste' 'spuntino' 'spuntini',
        before [; "Adesso non @`e il momento di pensare al CIBO."; ],
    has scenery proper;
```

```
Object  menu "menu" bar
    with
        name 'menu' 'lista' 'listino',
        description
            "Il menu appeso al muro elenca tutti i cibi e bevande che
             Benny pu@`o servire. Peccato che tu non abbia mai imparato
             a leggere, ma fortunatamente c'@`e il disegno di una grossa
             tazza di caff@`e fra tutte le altre scritte
             incomprensibili.",
        before [;
            Take:
                "Il menu @`e affisso al muro alle spalle di Benny.
                 Inoltre @`e inutile SCRITTURA.";
        ],
    has scenery;

Object  clienti "clienti" bar
    with
        name 'clienti' 'persone' 'cliene' 'gente' 'uomini' 'donne',
        description [;
            if (costume has worn)
                "La maggior parte sembra concentrarsi sul proprio cibo,
                 ma alcuni ti osservano blaterando. Deve essere colpa dei
                 colori IPNOTIZZANTI-INSTUPIDENTI del tuo costume.";
            else
                "Un groppo di INERMI e IGNARI mortali, gli stessi che
                 Capitan FATO ha giurato di DIFENDERE il giorno in cui i
                 suoi genitori si sono soffocati con una MALIGNA fetta di
                 TORTA DI MIRTILLI.";
        ],
        life [;
            Ask,Tell,Answer:
                if (costume has worn)
                    "La gente sembra NON FIDARSI dell'asspetto del tuo
                     FAVOLOSO costume.";
                else
                    "Come John Covarth, sei MENO interessante del cibo di
                     Benny.";
            Kiss:
                  "Non saprei dirti quali tipi di batteri MUTANTI questi
                   STRANIERI stanno portando.";
            Attack:
                "L'insensato massacro di civili @`e una caratteristica
                 dei CATTIVI. Si SUPPONE che tu protegga le persone come
                 queste.";
        ],
        orders [; "Queste persone non sembrano essere cooperative."; ],
        numero_di_commenti 0,    ! per contare i commenti dei clienti
        daemon [;
            if (location ~= bar) return;
            if (self.numero_di_commenti == 0) {
                self.numero_di_commenti = 1;
                print      "I clienti guardano il tuo costume con aperta
                            curiosità";
            }
            if (random(2) == 1) { ! esegui questo il 50% delle volte
                self.numero_di_commenti = self.numero_di_commenti + 1;
                switch (self.numero_di_commenti) {
                    2:      "^~Non sapevo ci fosse il circo in città,~
                             dice un clienti ad un altro. ~Sembra che i
                             pagliacci abbiano il giorno libero.~";
```

```
                3:      "^~Questi stilisti non sanno più che fare per
                         farsi    conoscere,~    sbuffa    un    signore
                         corpulento guardando nella tua direzione.
                         Quelli   che   hanno   sentito   cercano   di
                         nascondere il sogghigno.";
                4:      "^~Deve essere di nuovo carnevale,~ dice un
                         uomo a sua moglie, che sogghigna dandoti
                         un'occhiata di sfuggita. ~Come vola il
                         tempo...~";
                5:      "^~La cosa peggiore delle grandi città~,
                         commenta qualcuno parlando con il suo
                         compagno di tavolo, ~è che vedi gli insetti
                         più schifosi uscire dai cessi.~";
                6:      "^~VORREI davvero poter andare al lavoro in
                         pigiama,~dice una ragazza in tailleur ai suoi
                         colleghi. ~È COSÌ comodo.~";
                default: StopDaemon(self);
            }
        }
    ],
    has scenery animate pluralname;

Object  benny "Benny" bar
    with
        name 'benny',
        description
            "Un uomo ingannevolmente GRASSO dotato di incredibile
             agilit@`a, Benny intrattiene i clienti schiacciandosi noci
             di cocco sulla fronte quando @`e dell'umore giusto.",
        chiesto_caffe    false,      ! il giocare ha chiesto un caff@`e?
        caffe_non_pagato false,      ! Benny aspetta di essere pagato?
        chiave_non_resa  false,      ! Benny aspetta la chiave indietro?
        life [;
            Give:
                switch (noun) {
                    vestiti:
                        "Hai BISOGNO degli anonimi vestiti di John
                         Covarth.";
                    costume:
                        "Hai BISOGNO della tua stupenda tuta ANTI-
                         ACIDO.";
                    chiave_del_gabinetto:
                        self.chiave_non_resa = false;
                        move chiave_del_gabinetto to benny;
                        "Benny annuisce mentre tu MIRABILMENTE gli rendi
                         la chiave.";
                    moneta:
                        remove moneta;
                        self.caffe_non_pagato = false;
                        print  "Con meravigliose movenze da ILLUSIONISTA,
                                fai apparire una moneta dal tuo ";
                        if (costume has worn)
                            print "costume a PROVA DI PROIETTILE";
                        else print "normale vestito di tutti i giorni";
                        " come se fosse uscita fuori dall'orecchio di
                         Benny! Le persone attorno a te applaudono
                         educatamente. Benny prende la moneta, e la morde
                         SOSPETTOSO. ~Grazie, signore. Torni quando
                         vuole,~ dice.";
                }
```

```
    Attack:
        if (costume has worn) {
            deadflag = 4;
            print  "Davanti agli occhi pieni di orrore della
                    gente circostante, salti MAGNIFICIENTEMENTE
                    OLTRE il bancone e attacchi Benny con
                    RIMARCHEVOLE, anche se NON sufficiente
                    velocit@`a. Benny ti riceve con uno sleale
                    uppercut che spedisce la tua MASCELLA DI
                    GRANITO                          attraverso
                    tutto il locale.^^ ~Questi uomini in pigiama
                    pensano di potersela   prendere con gente
                    innocente,~     sbuffa     Benny,     mentre
                    la SPETTRALE mano dell'OSCURITA' cala sulla
                    tua vista e tu perdi conoscenza.";
        }
        else
            "Questo non @`e un atto che potrebbe compiere il MITE
             John Covarth.";
    Kiss:   "Non c'@`e tempo per INSENSATE infatuazioni.";
    Ask,Tell,Answer:
        "Benny @`e troppo occupato per mettersi a chiaccherare.";
],
orders [;
! gestisce CHIEDI LA CHIAVE A BENNY e BENNY, DAMMI LA CHIAVE
    Give:
        if (second ~= player or nothing)
               "Benny ti guarda stranito.";
        switch (noun) {
            chiave_del_gabinetto:
                if (chiave_del_gabinetto in player)
                    "Ma tu HAI già la chiave.";
                if (self.chiesto_caffe == true) {
                    if (chiave_del_gabinetto in self) {
                        move chiave_del_gabinetto to player;
                        self.chiave_non_resa = true;
                       "Benny getta la chiave delle toilettes sul
                        bancone, da cui tu la prendi con un destro
                        e preciso movimento della tua mano SUPER-
                        AGILE.";
                    }
                    else   "~L'ultimo posto in cui ho visto quella
                            chiave, è stata la TUA mano, ~ mugugna
                            Benny. ~Assicurati di restituirgliela
                            prima di andar via.~";
                }
                else   "~Il gabinetto è solo per i clienti.~
                        mormora, indicando con il dito il menu
                        alle sue spalle.";
            caffe:
                if (self.chiesto_caffe == true)
                    "Un caffè mi sembra abbastanza.";
                move caffe to bancone;
                self.chiesto_caffe = true;
                self.caffe_non_pagato = true;
                "In sole due mosse aggraziate, Benny posa di
                 fronte a te il suo famosissimo Caffè
                 macchiato.";
            cibo:
                "Mangiare ti prenderebbe troppo tempo, devi
                 cambiarti ORA.";
```

```
                menu:
                    "Con solo un piccolissimo singhiozzo, Benny fa un
                     cenno verso il menu affisso al muro alle sue
                     spalle.";
                default: "~Non credo sia sul menù, signore.~";
            }
        ],
    has scenery animate male proper transparent;

Object  caffe "tazza di caff@`e" benny
    with
        name   'tazza'  'di'  'caffe'  'caffe^'  'caff@`e'  'macchiato'
               'cappuccino',
        initial "Sul bancone, il caff@`e fumante ti st@`a aspettando.",
        description [;
            if (self in benny)
                "Il disegno sul men@`u ha SICURAMENTE un bell'aspetto.";
            else
                "Aroma delizioso.";
        ],
        before [;
            Take,Drink,Taste:
                if (self in benny)
                    "Forse dovresti ordinarne uno a Benny.";
                else {
                    move self to benny;
                    print  "Prendi la tazzina e ne bevi un sorso. La
                            REPUTAZIONE  MONDIALE  di  Benny  @`e  ben
                            meritata. Appena finisci, Benny porta via la
                            tazzina.   ~Il   caff@`e   viene   un'euro,
                            signore.~";
                    if (benny.caffe_non_pagato == true)
                           "~Viene un euro, Signore~";
                    else "" ;
                }
            Buy:
                if (moneta in player) <<Give moneta benny>>;
                else "Non hai soldi.";
            Smell:
                "Se la tua IPERATTIVA ghiandola pituitaria @`e
                 affidabile, @`e una miscela colombiana.";
        ],
    has female;

Object  fuori_dal_gabinetto "gabinetto" bar
    with
        name 'gabinetto' 'cesso' 'toilette' 'toilet' 'ritirata' 'bagno',
        before [;
            Enter:
                if (porta_del_gabinetto has open) {
                    PlayerTo(gabinetto);
                    return true;
                }
                else
                    "La tua SUPERBA mente deduttiva comprende che la
                     PORTA @`e chiusa.";
            Examine:
                if (porta_del_gabinetto has open)
                    "Un pensiero brillante illumina il tuo cervello
                     SUPERLATIVO: una dettagliata esplorazione del
                     gabinetto  sarebbe ESTREMAMENTE
```

```
                facilitata se tu entrassi all'interno.";
            else
                "Con un TREMENDO sforzo di volont@`a, evochi la tua
                 imperscrutabile VISIONE ASTRALE e la proietti in
                 AVANTI attraverso la porta chiusa... fino a che non
                 ti ricordi che @`e il Dottor Mystere ad avere i
                 poteri mistici.";
        Open:  <<Open  porta_del_gabinetto>>;
        Close: <<Close porta_del_gabinetto>>;
        Take,Push,Pull: "Sarebbe PARTE dell'edificio.";
    ],
has scenery openable enterable;

Object  porta_del_gabinetto
    with
        pname  'porta'  '.x'  'rossa'  '.x'  'del'  '.x'  'gabinetto'  '.x'
               'bagno',
        short_name [;
            if (location == bar)
                print "porta che conduce al gabinetto";
            else print "porta che conduce al bar";
            return true;
        ],
        description [;
            if (location == bar)
                "Una porta rossa con le inequivocabili silhouette di un
                 uomo e di una donna che segnano l'ingresso ai locali
                 igienici. C'@`e una nota scarabocchiata attaccata alla
                 superficie della porta.";
            else
                "Una porta rossa senza alcuna caratteristica NOTEVOLE.";
        ],
        found_in bar gabinetto,
        before [ ks;
            Open:
                if  (self hasnt locked || chiave_del_gabinetto notin
                    player)
                        return false;
                ks = keep_silent; keep_silent = true;
                <Unlock self chiave_del_gabinetto>; keep_silent = ks;
                return true;
            Lock:
                if (self hasnt open) return false;
                print "(prima chiudi ", (the) self, ")^";
                ks = keep_silent; keep_silent = true;
                <Close self>; keep_silent = ks;
                return false;
        ],
        after [ ks;
            Unlock:
                if (self has locked) return false;
                print "Apri ", (the) self, ".^";
                ks = keep_silent; keep_silent = true;
                <Open self>; keep_silent = ks;
                return true;
            Open:  give gabinetto light;
            Close: give gabinetto ~light;
        ],
        door_dir [;
            if (location == bar)  return n_to;
            else                  return s_to;
```

```
        ],
        door_to [;
            if (location == bar)  return gabinetto;
            else                  return bar;
        ],
        with_key chiave_del_gabinetto,
    has scenery door openable lockable locked female;

Object  chiave_del_gabinetto "chiave del gabinetto" benny
    with
        pname 'chiave' '.x' 'del' '.x' 'gabinetto' '.x' 'bagno',
        article "la",
        invent [;
            if (vestiti has worn) print "la chiave CRUCIALE";
            else print "la chiave ormai usata e IRRILEVANTE";
            return true;
        ],
        description
            "I  tuoi  sensi  ULTRA-PERCETTIVI  non  individuano  niente  di
            particolare sulla chiave del gabinetto.",
        before [;
            if (self in benny)
                "SCANDAGLI i l'ambiente con la tua CONSAPEVOLEZZA
                 POTENZIATA, ma non riesci ad individuare alcuna
                 chiave.";
            Drop:  "Benny si aspetta che tu restituisca la chiave prima
                    o poi.";
        ],
     has female;

Object  "nota" bar
    with
        name 'nota' 'scarabocchiata',
        description [;
            if (self.letta == false) {
                self.letta = true;
                    "Rivolgi la tua visione a ULTRAFREQUENZA AVANZATA
                     verso la nota e socchiudi gli occhi concentrandoti,
                     arrendendoti solo quando i bordi della nota iniziano
                     ad annerirsi sotto l'incredibile intensit@`a del tuo
                     sguardo infuocato. Rifletti ancora una volta su
                     quanto sarebbe stato utile se tu avessi mai imparato
                     a leggere. ^^Una vecchia signora premurosa si
                     avvicina e ti spiega: ~Devi chiedere la chiave a
                     Benny, al bancone.~^^ Ti volti verso di lei e inizi
                     a dire: ~Oh, la SAPEVO, ma...~^^ ~Di nulla,
                     figliolo,~ dice la signora mentre esce dal bar.";
            }
            else
                "La nota indecifrabile e annerita non ha pi@`u SEGRETI
                 per te ADESSO. Ha!";
        ],
        letta false,  ! il giocatore ha gi@`a letto la nota?
        before [;
            Take: "Non hai motivo di raccogliere note INDECIFRABILI.";
        ],
    has scenery female;

!-----------------------------------------------------------------------
```

```
Room    gabinetto "Un gabinetto unisex"
    with
        description
            "Una stanza quadrata, incredibilmente PULITA, dalle pareti
             ricoperte di mattonelle di ceramica, che non contiene molto
             di pi@`u di un gabinetto e un interruttore. L'unica uscita
             @`e a sud, attraverso la porta che riconduce al bar.",
        s_to porta_del_gabinetto,
    has ~light scored;

Appliance interruttore "interruttore" gabinetto
    with
        name 'interruttore' 'luce',
        description
            "Un notevole PRODIGIO di tecnologia e SCIENZA, elegante e
             FACILE da usare.",
        before [;
            Push:
                if (self has on) <<SwitchOff self>>;
                else             <<SwitchOn  self>>;
        ],
        after [;
            SwitchOn:
                give self light;
                "Accendi la luce del gabinetto.";
            SwitchOff:
                give self ~light;
                "Spegni la luce del gabinetto.";
        ],
    has switchable ~on;

Appliance cesso "gabinetto" gabinetto
    with
        name 'gabinetto' 'wc' 'cesso' 'water' 'water-closed' 'tazza',
        before [;
            Examine:
                if (moneta in self) {
                    move moneta to parent(self);
                    "L'ultima persona ha CIVILMENTE tirato l'acqua dopo
                     aver usato il gabinetto, ma ha dimenticato di
                     raccogliere la PREZIOSA moneta che @`e caduta dai
                     suoi pantaloni.";
                }
            Receive:
                "Mentre qualsiasi altro MORTALE potrebbe gettare
                 QUALSIASI cosa senza pensarci ", (artin) self, ", ti
                 ricordi i profondi insegnamenti del tuo mentore, il Duca
                 ELEGANTE, di come usare e cavalcare i cessi.";
        ];

Object  moneta "moneta" cesso
    with
        name 'moneta' 'euro' 'soldi',
        description "E' una moneta da un EURO.",
        before [;
            Drop:
                if (self notin player) return false;
                    "Lasciare una moneta tanto preziosa? Ha, ha! Questa
                     deve essere una dimostrazione del tuo ULTRA-FRIVOLO
                     senso dell'umorismo!";
        ],
```

```
        after [;
            Take:
                "Ti accosci nella posizione del DRAGO DORMIENTE e senza
                 indugio raccogli e intaschi la moneta con la PRESA
                 SOVRANA.";
        ],
    has scored female;

!=====================================================================
! Oggetti del giocatore

Object  vestiti "i tuoi vestiti"
    with  name 'vestiti' 'vestito' 'ordinari' 'abiti' 'abito',
        description
            "Vestiti   perfettamente   ORDINARI   per   un   NESSUNO   come
            John Covarth.",
        before [;
            Wear:
                if (self has worn) "Sei già vestito come John Covarth.";
                else   "La città ha BISOGNO dei poteri di capitan FATO,
                        non dell'anonimato di John Covarth.";
            Disrobe,Change:
                if (self hasnt worn)
                    "La tua vista ACUTA ti dice non stai più indossando",
                     (the) self, ".";
                switch (location) {
                    strada:
                        if (player in cabina)
                            "Non avendo la super-velocità di Superman,
                             realizzi che sarebbe sconveniente cambiarti
                             sotto gli occhi delle persone che passano.";
                        else
                            "In mezzo di strada? Questo sarebbe uno
                             SCANDALO, e inoltre rivelerebbe la tua
                             identità segreta.";
                    bar:
                        "Benny non sopporta le buffonate nel suo
                         locale.";
                    gabinetto:
                        if (porta_del_gabinetto has open)
                            "La porta rimane aperta, e ci sono decine di
                             occhi curiosi. Dovresti arrestarti da solo
                             per condotta IMMORALE.";
                        print  "   Ti togli rapidamente i vestiti e li
                                raccogni in un pacco ULTRA-MINUSCOLO
                                facilmente trasportabile. ";
                        if (porta_del_gabinetto has locked) {
                            give vestiti ~worn; give costume worn;
                            "Poi spieghi il tuo costume in COTONE
                             INVULNERABILE e ti trasformi in Capitan FATO,
                             difensore della libertà e avversario della
                             tirannia!";
                        }
                        else {
                            deadflag = 3;
                            "Stai per infilarti il costume di Capitan
                             FATO, quando la porta si apre e una giovane
                             donna entra. LEI ti guarda e inizia ad
                             urlare, ~UNO STUPRATORE. UNO STUPRATORE NUDO
                             NEL BAGNO!!!~^^ Tutti coloro che erano nel
                             bar giungono in soccorso, solo per vederti
```

```
                        saltare in modo ridicolo su una gamba sola
                        mentre cerchi di vestirti. Le loro risate
                        segnano la RAPIDA FINE della tua carriera di
                        combattente del crimine!";
                    }
                thedark:
                    "L'ultima volta che ti sei cambiato al buio, hai
                     indossato il costume al contrario!";
                default:  ! questo caso non dovrebbe mai capitare
                    "Ci sono posti migliori dove cambiarsi d'abito.";
            }
        ],
    has clothing proper pluralname;

Object  costume "il tuo costume"
    with
        name 'capitan' 'fato' 'costume' 'tuta',
        description
            "Manifattura allo STATO DELL'ARTE, 100% COTONELASTICO(tm)
             rinforzato chimicamente.",
        before [;
            Wear:
                if (self has worn) "Sei già vestito da Capitan Fato.";
                else   "Prima dovresti toglierti gli ordinari e ANONIMI
                        vestiti di John Covarth.";
            Disrobe,Change:
                if (self has worn)
                    "Hai BISOGNO del tuo costume per combattere il
                     crimine!";
                else "Non lo stai indossando!";
            Drop:
                "Il tuo UNICO costume multicolore da Capitan FATO? Il
                 pi@`u desiderato capo d'abbigliamento in tutta la
                 citt@`a? Certamente NO!";
        ],
    has clothing proper;

!=============================================================================
! Routine entry point

[ Initialise;
    #Ifdef DEBUG; pname_verify(); #Endif;       ! suggerito da pname.h
    location = strada;
    move costume to player;
    move vestiti to player; give vestiti worn;
    lookmode = 2;
    "^^Impersonando il tranquillo John Covarth, assistente garzone in
     una insignificante drogheria, ti FERMI di colpo quando il tuo
     udito finissimo decifra una chiamata radio della POLIZIA. Un FOLLE
     sta attaccando la popolazione al Parco Granaio! Devi indossare
     velocemente il tuo costume da Capitan FATO...!^^";
];

[ DeathMessage;
    if (deadflag == 3)
        print "La tua identit@`a segreta @`e stata rivelata.";
    if (deadflag == 4) print "Sei stato VERGOGNOSAMENTE sconfitto.";
    if (deadflag == 5)
        print "Voli via, diretto a RISOLVERE la SITUAZIONE!";
];
```

```
[ InScope person item;
    if (person == player && location == thedark && real_location ==
       gabinetto) {
           PlaceInScope(interruttore);
           PlaceInScope(porta_del_gabinetto);
    }
    if (person == player && location == thedark)
       objectloop (item in parent(player))
           if (item has moved) PlaceInScope(item);
           return false;
];

!=============================================================================
! Grammatica standard e estesa

Include "ItalianG";

[ ChangeSub;
    if (noun has pluralname) print "Non sono";
    else                     print "Non @`e";
    " qualcosa che devi cambiare per risolvere la situazione.";
];

Verb 'cambia'
    * noun                              -> Change;

Extend 'chiedi'
    *   'a'/'ad'/'all^'/'allo'/'alla'/'al'/'agli'/'ai'/
       'alle' creature topic            -> Askfor
    *   topic 'a'/'ad'/'all^'/'allo'/'alla'/'al'/'agli'/'ai'/
       'alle' creature
                                        -> AskFor reverse;

!=============================================================================
```

Compilare strada facendo

"Capitan Fato" presenta le stesse difficoltà di "Guglielmo Tell": se digitate il codice così come lo trovate mentre leggete la guida, l'avventura non potrà essere compilata fin quando non si raggiunge la fine del tredicesimo capitolo. Per compilare il gioco mentre si procede alla lettura dei capitoli precedenti è necessario aggiungere questi mozziconi alla fine del codice presentato al termine del Capitolo 10.

```
! ============================================================
! DEFINIZIONI TEMPORANEE NECESSARIE ALLA COMPILAZIONE ALLA FINE DEL
! CAPITOLO 10

Room bar;
Object vestiti;
```

Sostituite questi mozziconi con questi altri alla fine del Capitolo 11:

```
! ============================================================
! DEFINIZIONI TEMPORANEE NECESSARIE ALLA COMPILAZIONE ALLA FINE DEL
! CAPITOLO 11

Room bagno;
Object vestiti;
Object costume;
```

e con questi alla fine del Capitolo 12:

```
! ============================================================
! DEFINIZIONI TEMPORANEE NECESSARIE ALLA COMPILAZIONE ALLA FINE DEL
! CAPITOLO 12

Room bagno;
Object vestiti;
Object costume;
Object moneta;
Object caffe;
Object cibo;
Object menu;
```

Alla fine del capitolo 13 il gioco sarà completo, pertanto potrete eliminare i mozziconi temporanei.

Appendice E: Il linguaggio Inform

Fate riferimento a questa appendice per un riassunto succinto ma sostanzialmente completo del linguaggio di programmazione Inform; copre tutto quello che abbiamo trattato in questa guida, più vari costrutti che non sono capitati naturalmente, ed altri di natura avanzata od oscura.

Literal

Nel linguaggio specialistico dei computer l'unità fondamentale di immagazzinamento dati è un **byte** composto da otto bit, in grado di memorizzare i valori tra 0 e 255. È così piccolo che è utile a contenere niente altro che un singolo carattere, quindi la maggior parte dei computer lavora con gruppi di due, quattro od otto byte conosciuti come **word** [parola] (da non confondere con le voci del dizionario di Inform [dictionary word]). Nella Z-Machine una word di immagazzinamento è composta da due byte e potete specificare in varii modi il valore letterale che deve memorizzare.

- Decimale: da `-32768` a `32767`

  Esadecimale: da `$0` a `$FFFF`

  Binario: da `$$0` a `$$1111111111111111`

- Azione: `##Look`

- Carattere: `'a'`

- Voce di dizionario: 'maiale' (fino a nove caratteri significativi); usate l'accento circonflesso `"^"` per indicare l'apostrofo

  Parola plurale: `'maiali//p'`

  Parola di un carattere: `"e"` (solo nella proprietà name) o `"e//"`

- Stringa: `"l'avventura del maiale"` (per un massimo di 4000 caratteri circa); può includere valori speciali tra cui:

^	vai a nuova riga
~	virgolette "
`@@64`	il segno della chiocciola "@"
`@@92`	barra inversa (backslash) "\"
`@@94`	accento circonflesso "^"
`@@126`	tilde "~"
`@`a`	a con accento grave "à", e così via
`@LL`	il simbolo della sterlina "£", e così via
`@@00..@@31`	stringhe da `00` a `31`

Nomi

Sono gli identificatori di *`const_id`* [costanti], *`var_id`* [variabili], *`array`* [matrici], *`class_id`* [classi], *`obj_id`* [oggetti], *`property`* [proprietà], *`attribute`* [attributi], *`routine_id`* o *`label`* [etichette] di Inform. Accettano fino a 32 caratteri: alfabetici (maiuscole e minuscole non fanno differenza), numerici e l'underscore (_); il primo carattere non deve essere un numero.

Costanti

Valori word con un nome, che non cambiano durante l'esecuzione, e che sono sono inizializzati a zero in mancanza di altra indicazione:

```
Constant const_id;
Constant const_id = espressione;
```

Costanti standard sono `true` (1), `false` (0) e `nothing` (0), anche `NULL`(-1). In più, `WORDSIZE` è il numero di byte in una parola di immagazzinamento (word): 2 per la Z-machine e 4 per Glulx.

Per definire una costante (a meno che già esista):

```
Default const_id espressione;
```

Variabili ed array [matrici]

Valori byte/word con un nome che possono cambiare durante l'esecuzione e che sono inizializzati a zero in mancanza di altra indicazione:

Una variabile **globale** è una word singola:

```
Global var_id;
Global var_id = espressione;
```

Un **array di word** è un insieme di word globali a cui si accede con `array-->0, array-->1, ..., array-->(N-1)`:

```
Array array --> N;
Array array -->espr1 espr2...esprN;
Array array --> "stringa";
```

Un **array di tabelle** è un insieme di word globali a cui si accede con `array-->1, array-->2, ..., array-->N`, ed in cui `array-->0` è inizializzato a *N*:

```
Array array table N;
Array array table esp1 esp2...espN;
Array array table "stringa";
```

Un **array di byte** è un insieme di byte globali a cui si accede con `array->0, array->1, ..., array->(N-1)`:

```
Array array -> N;
Array array -> espr1 espr2...esprN;
Array array -> "stringa";
```

Un **array di stringhe** è un insieme di byte globali a cui si accede con `array->1, array->2, ..., array->N`, ed in cui `array->0` è inizializzato a *N*:

```
Array array string N;
Array array string espr1 ... esprN;
Array array string "stringa";
```

Un **array di buffer** è un insieme di byte globali a cui si accede usando `array->(WORDSIZE), array->(WORDSIZE+1) ... array->(WORDSIZE+N-1)`, con la prima **word** array->0 inizializzata a *N*:

In tutti questi casi i caratteri della stringa che inizializza sono estratti nei singoli elementi byte/word dell'array.

Consultate anche Oggetti (per le variabili di **proprietà**) e Routine (per le variabili **locali**).

Espressioni ed operatori

Per controllare l'ordine di valutazione si usano le parentesi `(...)`.

Le espressioni aritmetiche/logiche supportano questi operatori:

`p + q`	addizione
`p - q`	sottrazione
`p * q`	moltiplicazione
`p / q`	divisione intera
`p % q`	resto della divisione
`p++`	incrementa *p*, restituisce il valore originale
`++p`	incrementa *p*, restituisce il nuovo valore
`p--`	decrementa *p*, restituisce il valore originale
`--p`	decrementa *p*, restituisce il nuovo valore
`p & q`	**AND** a livello di bit
`p | q`	**OR** a livello di bit
`~p`	**NOT** a livello di bit (inversione)

Le espressioni condizionali restituiscono `true` [vero] o `false` [falso]; *q* può essere un elenco di possibilità *q1 or q2 or ... qN*:

`p == q`	*p* è uguale a *q*
`p ~= q`	*p* è diverso da *q*
`p > q`	*p* è più grande di *q*
`p < q`	*p* è più piccolo di *q*
`p >= q`	*p* è maggiore od uguale a *q*
`p <= q`	*p* è minore od uguale a *q*
`p ofclass q`	l'oggetto p è della classe *q*
`p in q`	l'oggetto *p* è figlio di *q*
`p notin q`	l'ogg. *p* non è figlio di *q*
`p provides q`	l'ogg. *p* fornisce la propr. *q*
`p has q`	l'oggetto *p* ha l'attributo *q*
`p hasnt q`	l'ogg. *p* non ha l'attributo *q*

Le espressioni booleane restituiscono `true` o `false`; se *p* è determinante per il risultato, *q* non viene valutata:

`p && q`	sia *p* che *q* sono `true` (non zero)
`p || q`	uno dei due è `true` (non zero)
`~~p`	*p* è `false` (zero)

Per restituire -1, 0 o 1 in base ad un confronto senza segno:

```
UnsignedCompare(p, q)
```

Per restituire `true` se l'oggetto *q* è figlio o nipote o... di *p*:

```
IndirectlyContains(p, q)
```

Per restituire il parente comune più vicino (o `nothing`) di due oggetti:

```
CommonAncestor(p, q)
```

Per restituire un numero casuale tra 1 e *N*, od uno tra una lista di valori costanti:

```
random(N)
random(valore, val, ..., valore)
```

Classi ed oggetti

Per dichiarare un *class_id* - un template [modello] per una famiglia di oggetti - dove il valore opzionale (*N*) limita il numero di istanze create durante l'esecuzione:

```
Class class_id(N)
class class_id ... class_id
with  prop_def,
      ...
      prop_def,
has   attr_def ... attr_def;
```

Per dichiarare un *obj_id*, "Object" può essere usato al posto di *class_id*, gli altri quattro elementi di intestazione sono opzionali e le frecce (->, -> ->, ...) e *parent_obj_id* sono incompatibili:

```
Object frecce obj_id "nome_esteso"
       parent_obj_id
class  class_id ... class_id
with   prop_def,
       ...
       prop_def,
has    attr_def ... attr_def;
```

I segmenti class, with e has (anche lo scarsamente usato private) sono opzionali e possono apparire in qualsiasi ordine.
Per determinare se la classe di un oggetto fa parte di Class, Object, Routine, String (o nothing):

```
metaclass(obj_id)
```

Segmento `has`: Ogni *attr_def* può essere:

```
attributo
~attributo
```

Per modificare gli attributi durante l'esecuzione:

```
give obj_id attr_def ... attr_def;
```

Segmenti `with/private`: Ogni *prop_def* dichiara una variabile (o un array di word) e può prendere una di queste forme (dove un valore è un'espressione, una stringa od una routine incorporata):

```
proprietà
proprietà valore
proprietà valore ... valore
```

Una variabile di proprietà viene indirizzata tramite *obj_id.proprietà* (od all'interno della dichiarazione dell'oggetto con *self.proprietà*). Valori multipli creano un array di proprietà; in questo caso *obj_id.#proprietà* è il numero di **byte** occupati dall'array, si può accedere ai singoli elementi con *obj_id.&proprietà-->0*, *obj_id.&proprietà-->1*, ..., e *obj_id.proprietà* si riferisce al valore del primo elemento.
Una variabile di proprietà ereditata da una classe oggetto si indirizza tramite *obj_id.class_id::proprietà* che restituisce il valore originale prima delle modifiche che vengono apportate all'interno dell'oggetto.

Manipolare l'albero degli Oggetti

Per modificare le relazioni tra oggetti durante l'esecuzione:

```
move obj_id to parent_obj_id;
remove obj_id;
```

Per ottenere il genitore di un oggetto (o nothing):

```
parent(obj_id)
```

Per ottenere il primo figlio di un oggetto (o nothing):

```
child(obj_id)
```

Per ottenere il figlio adiacente di un oggetto genitore (o nothing):

```
sibling(obj_id)
```

Per ottenere il numero di oggetti figli diretti di un oggetto:

```
children(obj_id)
```

Invio di messaggi

Ad una classe:

```
class_id.remaining()
class_id.create()
class_id.destroy(obj_id)
class_id.recreate(obj_id)
class_id.copy(da_obj_id, a_obj_id)
```

Ad un oggetto:

```
obj_id.proprietà(a1, a2, ... a7)
```

Ad una routine:

```
routine_id.call(a1, a2, ... a7)
```

Ad una stringa:

```
stringa.print()
stringa.print_to_array(array)
```

Istruzioni poco comuni e "deprecate"

Per saltare ad un'istruzione con etichetta:

```
jump etichetta;
...
.etichetta; istruzione;
```

Per terminare il programma:

```
quit;
```

Per salvare e ripristinare lo stato del programma:

```
save etichetta;
...
restore etichetta;
```

Per visualizzare il numero di versione del compilatore Inform:

```
inversion;
```

Per accettare dei dati dal flusso corrente di input:

```
read txt_array parse_array rout_id;
```

Per assegnare un valore ad una delle 32 variabili "low string":

```
string N "stringa";
Lowstring var_stringa "stringa";
string N var_stringa;
```

Istruzioni

Ogni istruzione deve terminare con un punto e virgola ";". Un *blocco_istruzioni* è un'istruzione singola od una serie di istruzioni racchiuse tra parentesi graffe {...}. Un punto esclamativo "!" segna l'inizio di un commento - il resto della riga viene ignorato.

Un'istruzione comune è quella di assegnazione:

```
var_id = espressione;
```

Ci sono due modi per fare un'assegnazione multipla:

```
var_id = var_id = ... = espr;
var_id = espr, var_id = espr, ... ;
```

Routine

Una routine può avere fino a 15 **variabili locali**: valori word che sono privati all'interno della routine (non visibili all'esterno) e che vengono impostati a 0 in mancanza d'indicazione ad ogni chiamata. La ricorsione è consentita.

Una routine **indipendente**:

- ha un nome con cui viene chiamata tramite *routine_id();* può anche essere chiamata indirettamente usando `indirect (routine_id,a1,a2,...,a7)`
- accetta argomenti, con *routine_id(a1, a2, ..., a7)* i cui valori inizializzano le variabili locali equivalenti
- restituisce `true` all'ultimo "]"

```
[ routine_id
  var_locale v_loc ... v_loc;
  istruzione;
  istruzione;
  ...
  istruzione;
];
```

Una routine **incorporata** come valore di una proprietà di un oggetto:

- non ha nome e viene chiamata quando la proprietà viene invocata; può anche essere chiamata esplicitamente usando *obj_id.proprietà*()
- accetta argomenti solo quando viene chiamata in maniera esplicita
- restituisce `false` all'ultimo "]"

```
 proprietà [
   var_locale v_loc ... v_loc;
   istruzione;
   istruzione;
   ...
   istruzione;
 ]
```

Le routine restituiscono un valore singolo quando l'esecuzione raggiunge l'ultimo "]" o quando viene incontrata un'istruzione `return`:

```
return espressione;
return;
rtrue;
rfalse;
```

Controllo del flusso

Per eseguire le istruzioni se l'espressione è vera (`true`); opzionalmente per eseguire altre istruzioni se l'espressione è falsa (`false`):

```
if (espressione)
    blocco_istruzioni
if (espressione)
    blocco_istruzioni
else
    blocco_istruzioni
```

Per eseguire le istruzioni in base al valore di espressione:

```
switch (espressione) {
    valore: istruz; ... istruz;
    valore: istruz; ... istruz;
    ...
    default: istruz; ... istruz;
}
```

dove ogni valore può essere fornito come:

```
costante
costante_infer to costante_super
costante, costante, ..., costante
```

E, se proprio dovete:

```
jump etichetta;
...
.etichetta; istruzione;
```

Cicli

Per eseguire le istruzioni mentre l'espressione è true:

```
while (espressione)
    blocco_istruzioni
```

Per eseguire le istruzioni fino a quando l'espressione diventa true:

```
do
    blocco_istruzioni
until (espressione)
```

Per eseguire le istruzioni mentre una variabile cambia:

```
for(imposta_var : ciclo_mentre_var :
aggiorna_var)
    blocco_istruzioni
```

Per eseguire le istruzioni per tutti gli oggetti definiti:

```
objectloop (var_id)
    blocco_istruzioni
```

Per eseguire le istruzioni per tutti gli oggetti selezionati da espressione:

```
objectloop (espr_comincia_con_var)
    blocco_istruzioni
```

Per uscire dal ciclo corrente più interno o dalla switch:

```
break;
```

Per cominciare immediatamente la successiva iterazione del ciclo corrente:

```
continue;
```

Visualizzare informazioni

Per visualizzare una lista di valori:
`print` *`valore, val, ..., val`*`;`

Per visualizzare una lista di valori seguita da un ritorno a capo e dalla restituzione del valore `true` dalla routine corrente:
`print_ret` *`valore, val, ..., val`*`;`

Se il primo (o l'unico) *`valore`* è una stringa, `print_ret` può essere omesso:
`"`*`stringa`*`", `*`valore, ..., valore`*`;`

Ogni *`valore`* può essere un'espressione, una stringa od una regola.

Un'**espressione** viene visualizzata come un valore decimale con segno.

Una **stringa** tra virgolette `"..."` viene visualizzata come testo.

Una **regola** è una tra:

`(number)`*`espr`* l'*`espr`* in parole

`(char)`*`espr`* l'*`espr`* è un carattere singolo

`(stringa)`*`indir`* la stringa che si trova all'indirizzo

`(address)` *`indir`* la voce di dizionario che si trova all'indirizzo

`(name)` *`obj_id`* il nome esterno (sintetico) dell'oggetto *`obj_id`*

`(a)` *`obj_id`* il nome dell'oggetto, preceduto da "un" "una" "uno" "un'" o "alcune" "alcuni" a seconda di genere e numero. Per i nomi propri mostra solo il nome.

`(the)` *`obj_id`* il nome dell'oggetto preceduto da "il" "lo" "la" "i" "gli" "le" "l'" a seconda di genere e numero.

`(The)` *`obj_id`* il nome dell'oggetto preceduto da "Il" "Lo" "La" "I" "Gli" "Le" a seconda di genere e numero.

`(thatorthose)` *`obj_id`*
stampa "quello" "quella" "quelli" "quelle" a seconda di genere e numero dell'oggetto.

`(cthetorthose)` *`obj_id`*
stampa "Quello" "Quella" "Quelli" "Quelle" a seconda di genere e numero dell'oggetto.

`(itorthem)` *`obj_id`*
stampa "lo" "la" "li" "le" a seconda di genere e numero dell'oggetto.

`(isorare)` *`obj_id`*
stampa "sono" "è" a seconda del numero dell'oggetto.

`(cisorare)` *`obj_id`*
stampa "Sono" "È" a seconda del numero dell'oggetto.

`(WhomorWhich1)` *`obj_id`*
stampa "lle quali" "i quali" "lla quale" "l quale" a seconda di genere e numero dell'oggetto.

`(whomorwhich2)` *`obj_id`*
stampa "il quale" "la quale" "i quali" "le quali" a seconda di genere e numero dell'oggetto.

`(artda)` *`obj_id`* stampa la preposizione articolata "da" più l'articolo, ad esempio "dallo", "dalla", etc. a seconda di genere e numero dell'oggetto.

`(artsu)` *`obj_id`* stampa la preposizione articolata "su" più l'articolo, ad esempio "sullo", "sulla", etc.

`(artin)` *`obj_id`* stampa la preposizione articolata "in" più l'articolo, ad esempio "nello", "nella", etc.

`(arta)` *`obj_id`* stampa la preposizione articolata "a" più l'articolo, ad esempio "allo", "alla", etc.

`(artdi)` *`obj_id`* stampa la preposizione articolata "di" più l'articolo, ad esempio "dello", "della", etc.

`(`*`routine_id`*`)`*`valore`*
l'output della chiamata *`routine_id`*`(`*`valore`*`)`

Per visualizzare un carattere di ritorno a capo:
`new_line;`

Per visualizzare degli spazi multipli:
`space` *`espressione`*`;`

Per visualizzare il testo in un box:
`box "`*`stringa`*`" "`*`stringa`*`" ... "`*`stringa`*`";`

Per cambiare dal set di caratteri normale a quello a spaziatura fissa:

```
font off;
...
font on;
```

Per cambiare gli attributi del set di caratteri:

```
style bold;      ! grassetto
style underline;! sottolineato
style reverse;   ! inversione
style roman;     ! normale
```

Verbi ed azioni

Per specificare un nuovo verbo:

```
Verb 'verbo' 'verbo' ... 'verbo'
* chiave chiave ... chiave -> azione
* chiave chiave ... chiave -> azione
...
* chiave chiave ... chiave -> azione
```

dove al posto di "`Verb`" si può usare "`Verb meta`", "*`azione`*" può essere "*`azione`* `reverse`", le chiavi sono opzionali ed ognuna può essere:

`'parola'`	quella parola
`'p1'/'p2'/...`	una di quelle parole
`attributo`	un oggetto con quell'attributo
`creature`	un oggetto che ha l'attributo animate
`held`	un oggetto in possesso del giocatore
`noun`	un oggetto *in scope* (a portata del giocatore)
`noun=`*`routine_id`*	un oggetto per il quale *`routine_id`* restituisca `true`
`scope=`*`routine_id`*	un oggetto in questa ridefinizione di scope
`multiheld`	uno o più oggetti in possesso del giocatore
`multi`	uno o più oggetti in scope
`multiexcept`	come multi, tranne l'oggetto indicato
`multiinside`	come multi, tranni quelli contenuti nell'oggetto specificato
`topic`	qualunque testo
`number`	qualunque numero
`routine_id`	una routine generale di parsing

Per aggiungere sinonimi ad un verbo esistente:

```
Verb 'verbo' 'verbo'... =
    'verbo_esistente';
```

Per modificare un verbo esistente:

```
Extend 'verbo_esistente' last
* chiave chiave ... chiave -> azione
* chiave chiave ... chiave -> azione
...
* chiave chiave ... chiave -> azione
```

dove al posto di "`Extend`" si può usare "`Extend only`" e "`last`" [ultimo] può essere omesso, o cambiato in "`first`" [primo] o "`replace`" [sostituisci].

Per eseguire esplicitamente un'azione (con *`noun`* e *`second`* opzionali, in base a quale azione si esegue):

```
<azione noun second>;
```

Per eseguire esplicitamente un'azione e restituire `true` dalla routine corrente:

```
<<azione noun second>>;
```

Altre direttive utili

Per impostare gli switch del compilatore inerite nelle *primissime linee* del file sorgente:

```
!% lista_switch_compilatore;
```

Per includere una direttiva all'interno della definizione di una routine `[...]`, inserite un simbolo del cancelletto "`#`" come primo carattere.

Per la compilazione condizionale:

```
Ifdef nome;       !se è definito nome
Ifndef nome;      !se non è definito
                  !nome
Iftrue espr;      !se l'espr è true
Iffalse espr;     !se l'espr è false
    ...
Ifnot;            !altrimenti
    ...
Endif;            !fine controllo
```

Per visualizzare un messaggio durante la compilazione:

```
Message "stringa";
```

Per includere il contenuto di un file, cercandolo all'interno del percorso Library (solitamente il percorso della libreria):

```
Include "codice_sorgente";
```

Per includere il contenuto di un file, cercandolo nella stessa directory in cui si trova il file corrente:

```
Include ">codice_sorgente";
```

Per indicare che una routine della libreria deve essere sostituita:

```
Replace routine_id;
```

Per impostare il numero di versione del gioco (l'impostazione predefinita è 1), il numero seriale (il predefinito è la data di oggi nel formato *aammgg*) ed il formato della riga di stato (il predefinito è `score`):

```
Release espr;          !la versione

Serial "aammgg";       !il numero
                       !seriale

Statusline score;      !riga di stato
                       !con punteggio

Statusline time;       !riga di stato
                       !con l'ora
```

Per dichiarare un nuovo attributo comune a tutti gli oggetti:

```
Attribute attributo;
```

Per dichiarare una nuova proprietà comune a tutti gli oggetti:

```
Property proprietà;
Property proprietà espressione;
```

Direttive poco comuni e "deprecate"

È improbabile che ne abbiate bisogno; cercatele all'interno dell'*Inform Designer's Manual* se necessario.

```
Abbreviate "stringa" ... "stringa";

End;

Import var_id var_id ... var_id;

Link "file_compilato";

Stub routine_id N;

Switches elenco__switch_compilatore

System_file;
```

Appendice F: La libreria di Inform

I file della libreria definiscono il modello del mondo (word model) di Inform, e sono quelli che trasformano un normale linguaggio di programmazione in un sistema di sviluppo per avventure testuali. Qui trovate le costanti, le variabili e le routine della libreria, le proprietà e gli attributi standard degli oggetti, la grammatica dei verbi e le azioni. Quest'ultima farà riferimento alla libreria italiana INFIT versione 2.5

Gli oggetti della libreria

`compass` [bussola]
: Un oggetto `container` [contenitore] che contiene i dodici oggetti di direzione `d_obj e_obj in_obj n_obj ne_obj nw_obj out_obj s_obj se_obj sw_obj u_obj w_obj`.

`LibraryMessages`
Se definito (tra gli `Include` di `Parser` e `Verblib`), cambia i messaggi standard della libreria:

```
Object LibraryMessages
   with before [;
       azione: "stringa";
       azione: "stringa";
       azione: switch (lm_n) {
                  valore: "stringa";
                  valore: "stringa",
                  (a) lm_o, ".";
               ...
           }
       ...
   ];
```

selfobj
: L'oggetto giocatore predefinito. Da evitare: al suo posto va usata la variabile `player`, che di solito si riferisce a `selfobj`.

thedark
: Una pseudo stanza che diventa la `location` quando non c'è luce (sebbene l'oggetto giocatore non vi venga trasferito).

Costanti della libreria

Oltre alle costanti standard `true` (1), `false` (0) e `nothing` (0) la Libreria definisce NULL (-1) per un *`action`*, *`property`* o *`pronoun`* il cui valore è indefinito.

`LIBRARY_PARSER`, `LIBRARY_ENGLISH`, `LIBRARY_VERBLIB` e `LIBRARY_GRAMMAR` sono definite per segnare la fine dell'inclusione di `Parser.h`, `English.h`, `VerbLib.h` e `Grammar.h` (o `ItalianG.h`) rispettivamente.

Se dichiarate un "`Constant PROMPT`" prima dell'inclusione dei file di libreria la vostra avventura userà un prompt alternativo al classico ">". Il parser porrà al giocatore domande del tipo "Cosa vuoi fare ora?", ecc. La procedura che gestisce questo tipo di prompt stampa in maniera (pseudo-)casuale un messaggio scelto tra quindici.

Costanti definite dall'utente

Alcune costanti controllano delle funzionalità invece di rappresentare dei valori.

`AMUSING_PROVIDED`
: Attiva l'entry-point Amusing.

`COMMENT_CHARACTER = '`*`carattere`*`'`
: Introduce una linea di commento ignorata dal parser, utile in modalità registrazione per il betatesting (predefinito "*").

`DEATH_MENTION_UNDO`
: Offre l'opzione "ANNULLARE il tuo ultimo comando" alla fine del gioco.

`DEBUG`
: Attiva i comandi di debug.

`Headline = "`*`stringa`*`"`
: **Obbligatoria**: stile del gioco, informazioni di copyright, ecc.

`MANUAL_PRONOUNS`
: I pronomi riflettono solo gli oggetti menzionati dal giocatore.

`MAX_CARRIED = `*`espressione`*
: Limite sul possesso diretto di cose che il giocatore può portare (predefinito 100).

`MAX_SCORE = `*`espressione`*
: Massimo punteggio del gioco (predefinito 0).

`MAX_TIMERS = `*`espressione`*
: Il limite di timer/daemon attivi (predefinito 32).

`NO_PLACES`
: Impedisce l'uso dei comandi "OGGETTI" e "POSTI".

`NO_SCORE`
: Il gioco non fa uso di punteggi.

`NUMBER_TASKS = espressione`
Numero di compiti con punteggio (predefinito 1).

`OBJECT_SCORE = espressione`
Quando viene preso un oggetto con punteggio la prima volta (predefinito 4).

`ROOM_SCORE = espressione`
Quando viene visitata una stanza con punteggio la prima volta (predefinito 5).

`SACK_OBJECT = obj_id`
Un oggetto `container` dove il gioco mette gli oggetti posseduti.

`START_MOVE = expr`
Valore iniziale del numero di turni (predefinito 0).

`Story = "stringa"`
Obbligatoria: il nome della storia.

`TASKS_PROVIDED`
Attiva il sistema di punteggio basato sui compiti da assolvere.

`USE_MODULES`
Attiva il link con i moduli di libreria precompilati.

`WITHOUT_DIRECTIONS`
Disattiva le direzioni standard della bussola (tranne "DENTRO" e "FUORI"). Mettere delle direzioni alternative in compass.

Variabili della libreria

`action`
L'azione corrente.

`actor`
L'obiettivo di un'istruzione: il giocatore, od un PNG.

`deadflag`
Di solito 0: 1 indica una morte normale, 2 indica che il giocatore ha vinto, 3 od altro indica una fine definita dall'utente.

`inventory_stage`
Usata dalle proprietà `invent` e `list_together`.

`keep_silent`
Di solito `false`; impostata a `true` rende la maggior parte delle azioni del gruppo 2 "silenziose".

`location`
La stanza in cui si trova attualmente il giocatore; a meno che sia buio, nel qual caso contiene `thedark`, mentre `real_location` contiene la stanza.

`notify_mode`
Normalmente `true`: impostata a `false` non vengono mostrati avvertimenti quando il punteggio cambia.

`noun`
L'oggetto primario a cui si riferisce l'azione corrente.

`player`
L'oggetto che agisce per conto del giocatore umano.

`real_location`
La stanza in cui si trova il giocatore quando è buio.

`score`
Il punteggio corrente.

`second`
L'oggetto secondario a cui si riferisce l'azione corrente.

`self`
L'oggetto che ha ricevuto il messaggio. (Nota: è una variabile a tempo d'esecuzione, non una costante a tempo di compilazione).

`sender`
L'oggetto che ha inviato un messaggio (o `nothing`).

`task_scores`
Un array di byte che contiene i punteggi per il sistema di punteggi basato sui compiti da assolvere.

`the_time`
L'orologio del gioco, in minuti (da 0 a 1439) trascorsi dalla mezzanotte.

`turns`
Il contatore delle mosse.

`wn`
Il numero di parola dal flusso di input, il conteggio inizia con 1.

Le routine della libreria

`Achieved(espressione)`
Un compito con punteggio è stato assolto.

`AfterRoutines()`
In un'azione del gruppo 2, controlla l'output dei messaggi `"after"` [dopo l'azione].

`AllowPushDir()`
Un oggetto può essere spinto da una stanza ad un'altra.

`Banner()`
Stampa il banner (l'intestazione) del gioco.

`ChangePlayer(obj_id, flag)`
Il giocatore diventa l'oggetto *obj_id*. Se il *flag* opzionale è *true*, la descrizione della stanza include `"(come oggetto)"`.

`CommonAncestor(obj_id1, obj_id2)`
Restituisce l'oggetto più prossimo che ha una relazione di parentela con entrambi gli oggetti indicati, o `nothing`.

`DictionaryLookup(array_di_byte, lunghezza)`
Restituisce l'indirizzo della parola nel dizionario, o 0 se non la trova.

`DrawStatusLine()`
Aggiorna la riga di stato; accade comunque alla fine di ciascun turno.

`GetGNAOfObject(obj_id)`
Restituisce la combinazione GNA (gender-number-animation) [sesso-numero-animato] con i valori da 0 a 11 dell'oggetto *obj_id*.

`HasLightSource(obj_id)`
Restituisce *true* se l'oggetto *obj_id* ha luce.

`IndirectlyContains(obj_id_genitore, obj_id)`
Restituisce `true` se `obj_id` è attualmente figlio, o nipote, o bis-nipote... dell'oggetto *obj_id_genitore*.

`IsSeeThrough(obj_id)`
Restituisce `true` se la luce passa attraverso *obj_id*.

`Locale(obj_id,"stringa1","stringa2")`
Descrive il contenuto di *obj_id* e ne restituisce il numero. Dopo gli oggetti con i propri paragrafi, il resto è elencato preceduto da *stringa1* o *stringa2*.

`LoopOverScope(routine_id, attore)`
Chiama *routine_id(obj_id)* per ogni *obj_id in scope*. Se il parametro opzionale *attore* viene fornito è quello che definisce lo *scope*.

`MoveFloatingObjects()`
Sistema la posizione degli oggetti `found_in`.

`NextWord()`
Restituisce la successiva parola del dizionaro dal flusso di input, incrementando `wn` di uno. Restituisce `false` se la parola non è nel dizionario, o se il flusso di input è terminato.

`NextWordStopped()`
Restituisce la successiva parola del dizionaro dal flusso di input, incrementando `wn` di uno. Restituisce `false` se la parola non è nel dizionario, -1 se il flusso di input è terminato.

`NounDomain(obj_id1, obj_id2, tipo)`
Effettua il parsing dell'oggetto (attiva l'analizzatore sintattico sull'oggetto); consultate anche `ParseToken()`.

`ObjectIsUntouchable(obj_id, flag)`
Verifica che ci sia una barriera (un oggetto `container` che non è aperto) tra il giocatore e l'oggetto *obj_id*. A meno che il *flag* opzionale sia `true` viene visualizzato il messaggio "Non puoi, ... in quella direzione.". Restituisce `true` se viene trovata una barriera, altrimenti restituisce `false`.

`OffersLight(obj_id)`
Restituisce `true` se l'oggetto *obj_id* fornisce luce.

`ParseToken(tipo, valore)`
Effettua un parsing normale; consultate anche `NoundDomain()`.

`PlaceInScope(obj_id)`
Viene usato in una proprietà `add_to_scope` oppure in `scope`=chiave per mettere l'oggetto *obj_id* in scope per il parser.

`PlayerTo(obj_id, flag)`
Muove il giocatore in *obj_id*. Stampa la descrizione a meno che il parametro opzionale *flag* sia 1 (nessuna descrizione) o 2 (come se ci fosse entrato).

`PrintOrRun(obj_id, proprietà, flag)`
Se *obj_id.proprietà* è una stringa la visualizza (seguita da un carattere di nuova riga a meno che il parametro opzionale *flag* sia `true`) e restituisce `true`. Se è una routine, la esegue e restituisce quello che la routine restituisce.

`PronounNotice(obj_id)`
Associa un pronome appropriato all'oggetto *obj_id*.

`PronounValue('pronome')`
Restituisce l'oggetto al quale 'esso' (o 'lui', 'lei', 'loro') si riferisce o `nothing`.

`ScopeWithin(obj_id)`
Viene usato in una proprietà `add_to_scope` oppure in `scope`=chiave per mettere il contenuto dell'oggetto *obj_id* in scope per il parser.

`SetPronoun('pronome', obj_id)`
Definisce l'oggetto *obj_id* al quale il pronome passato come parametro deve riferirsi.

`SetTime(espressione1, espressione2)`
Imposta `the_time` ad *espressione1* (in minuti trascorsi dalla mezzanotte, da 0 a 1439), e lo aggiorna con frequenza *espressione2*, secondo lo schema:
+espressione2: ad ogni turno scorrono *espressione2* minuti;
-espressione2: ogni minuto richiede *espressione2* turni;
0: il tempo non avanza.

`StartDaemon(obj_id)`
Avvia il `daemon` dell'oggetto *`obj_id`*.

`StartTimer(obj_id, espressione)`
Avvia il timer dell'oggetto *`obj_id`*, inizializzando il suo `time_left` [tempo rimanente] ad *`espressione`*. La proprietà `time_out` dell'oggetto viene chiamata dopo che il numero di turni è trascorso.

`StopDaemon(obj_id)`
Ferma il `daemon` dell'oggetto *`obj_id`*.

`StopTimer(obj_id)`
Ferma il timer dell'oggetto *`obj_id`*.

TestScope(*`obj_id, attore`*)
Restituisce `true` se l'oggetto *`obj_id è in scope`*, altrimenti restituisce `false`. Se il parametro opzionale *`attore`* viene fornito, allora è quello che definisce lo scope.

`TryNumber(espressione)`
Interpreta la parola *`espressione`* del flusso di input come se fosse un numero, riconoscendo i decimali, le parole inglesi da one a twenty, e se si stanno usando le librerie in italiano anche le parole in italiano da uno a venti. Restituisce i numeri da 1 a 10000, o -1000 se l'interpretazione va male.

`UnsignedCompare(espr1, espr2)`
Restituisce -1 se *`espr1`* è minore di *`espr2`*, 0 se *`espr1`* è uguale ad *`espr2`* ed 1 se *`espr1`* è maggiore di *`espr2`*. Entrambe le espressioni sono senza segno e variano da 1 a 65535.

`WordAddress(espressione)`
Restituisce un array di byte che contiene il testo della parola numero *`espressione`* dal flusso di input.

`WordInProperty(parola, obj_id, proprietà)`
Restituisce `true` se la parola del dizionario passata come parametro è presente tra i valori di proprietà dell'oggetto *`obj_id`*.

`WordLength(espressione)`
Restitusce la lunghezza della parola *`espressione`* dal flusso di input.

`WriteListFrom(obj_id, espressione)`
Visualizza un elenco con l'oggetto *`obj_id`* ed i suoi fratelli, nello stile fornito in *`espressione`* come la somma di:
`ALWAYS_BIT, CONCEAL_BIT, DEFART_BIT, ENGLISH_BIT, FULLINV_BIT, INDENT_BIT, ISARE_BIT, NEWLINE_BIT, PARTINV_BIT, RECURSE_BIT, TERSE_BIT, WORKFLAG_BIT.`

`YesOrNo()`
Restituisce `true` se il giocatore ha digitato "YES", restituisce `false` per "NO". Con le librerie italiane restituisce `true` se il giocatore ha digitato "SI" o "SÌ".

`ZRegion(argomento)`
Restituisce il tipo di *`argomento`*: 3 per l'indirizzo di una stringa, 2 per l'indirizzo di una routine, 1 per il numero di un oggetto, 0 per altro.

Le proprietà degli oggetti

Dove il *`valore`* di una proprietà può essere una routine è possibile usare diversi formati (ricordate, però, che le parentesi "`]`" delle routine incorporate restituiscono `false`, le parentesi "`]`" delle routine indipendenti restituiscono `true`):

`proprietà` `[; istruz; istruz; ... ]`

`proprietà` `[; return` *`routine_id`*`(); ]`

`proprietà` `[;` *`routine_id`*`(); ]`

`proprietà routine_id`

In questa appendice "♀" indica una proprietà additiva. Quando una classe (`Class`) ed un oggetto (`Object`) di quella classe definiscono la stessa proprietà, il valore specificato per l'oggetto normalmente sovrascrive il valore ereditato dalla classe. Quando una proprietà è additiva, invece, vengono applicati entrambi i valori, con quello dell'oggetto considerato per primo.

`add_to_scope`
Per un oggetto: altri oggetti che lo seguono in scope e fuori dallo scope. Il valore può essere un elenco di *`obj_id`* separati da spazi, od una routine che richiama `PlaceInScope()` o `ScopeWithin()` per specificare gli oggetti.

`after` ♀
Per un oggetto: riceve ogni *`action`* e *`fake_action`* per la quale questo è il `noun`. Per una stanza: riceve ogni *`action`* che accade nella stessa.
Il *`valore`* è una routine con una struttura simile a quella del costrutto `switch`, con una serie di casi per le *`action`* considerate (c'è anche un `default` opzionale); viene invocata dopo che l'azione è accaduta, ma prima che il giocatore venga informato. La routine deve restituire `false` per continuare, dicendo così al giocatore quello che è accaduto, o `true` per interrompere l'esecuzione dell'azione e non produrre ulteriori output.

`article`
Per un oggetto: l'articolo indeterminativo dell'oggetto. Il valore predefinito è automaticamente "a", "an", o "some". Usando le librerie italiane si hanno gli articoli indeterminativi italiani "un", "uno", "una", "degli", "delle", ecc. Il valore può essere una stringa od una routine che stampi una stringa.

`articles`
Per un oggetto non inglese: i suoi articoli determinativi ed indeterminativi. Il valore è un array di stringhe.

`before` ♀
Per un oggetto: riceve ogni *`action`* e *`fake_action`* e per la quale questo è il `noun`. Per una stanza: riceve ogni `action` che accade nella stessa.
Il *`valore`* è una routine richiamata prima che l'azione accada. Vedere anche la proprietà `after`.

`cant_go`
Per una stanza: il messaggio che viene stampato quando un giocatore cerca di usare un'uscita impossibile. Il valore può essere una stringa od una routine che stampi una stringa.

`capacity`
Per un oggetto `container` o `supporter`: il numero di oggetti che può contenere o che possono essere messi sopra, il valore predefinito è 100.
Per il giocatore: il numero di oggetti che può essere portato. `selfobj` ha una capacità iniziale di `MAX_CARRIED`.
Il *`valore`* può essere un numero od una routine che restituisca un numero.

d_to
Per una stanza: una possibile uscita (giù). Il *`valore`* può essere:
- `false` (predefinito): non è un'uscita;
- una *`stringa`*: viene stampata per spiegare perché questa non è un'uscita;
- una *`stanza`*: l'uscita porta a quella stanza;
- un *`oggetto door`* [porta]: l'uscita passa attraverso quella porta;
- una routine che deve restituire `false`, una *`stringa`*, una *`stanza`*, un oggetto *`door`*, oppure `true` per indicare una "non uscita" e non produrre altri output.

`daemon`
Il *`valore`* è una routine che può essere attivata da `StartDaemon(`*`obj_id`*`)` e che quindi viene eseguita ad ogni turno fino a che non viene disattivata da `StopDaemon(`*`obj_id`*`)`.

`describe` ♀
Per un *`oggetto`*: viene chiamata prima che venga stampata la descrizione dell'oggetto.
Per una *`stanza`*: viene chiamata prima che venga stampata la descrizione estesa della stanza.
Il *`valore`* è una routine che deve restituire `false` per continuare, stampando la descrizione normale, o `true` per interrompere l'azione e non produrre altri output.

`description`
Per un *`oggetto`*: la sua descrizione (visualizzata da `Examine`, `Esamina`).
Per una *`stanza`*: la sua descrizione estesa (Visualizzata da `Look`, `Guarda`).
Il *`valore`* può essere una stringa od una routine che stampi una stringa.

`door_dir`
Per un oggetto `compass` [bussola] (*`d_obj`*, *`e_obj`*, ...): la direzione nella quale si viene condotti quando si fa un tentativo di muoversi verso quest'oggetto.
Per un oggetto `door` [porta]: la direzione nella quale questo oggetto conduce.
Il *`valore`* può essere una proprietà di direzione (*`d_to`*, *`e_to`*, ...) od una routine che restituisca una tale proprietà.

`door_to`
Per un oggetto `door` [porta]: dove conduce. Il *`valore`* può essere
- *`false`* (il predefinito): non conduce da nessuna parte;
- una *`stringa`*: viene stampata per spiegare perché questa porta non conduce da nessuna parte;
- una *`stanza`*: la porta conduce in questa stanza;
- una routine che deve restituire `false`, una *`stringa`*, una *`stanza`*, oppure `true` per indicare che "non conduce da nessuna parte" e non produrre altri output.

`e_to`
Vedere `d_to`.

`each_turn` ♀
Viene invocata alla fine di ogni turno (dopo tutti i daemon ed i timer) ogni volta che l'oggetto è in scope. Il *`valore`* può essere una stringa od una routine.

`found_in`
Per un oggetto: le stanze nelle quali questo oggetto può essere trovato, a meno che abbia l'attributo `absent` [assente]. Il *`valore`* può essere:
- un elenco di stanze separate da spazi (dove questo oggetto può essere trovato) od un elenco di *`obj_id`* (le cui locazioni sono seguite da questo oggetto);
- una routine che deve restituire `true` se questo oggetto si può trovare nella locazione corrente, altrimenti deve restituire `false`.

`grammar`
Per un oggetto `animate` [animato] o `talkable` [a cui si può parlare]: il *`valore`*

è una routine chiamata quando il parser sa che si sta indirizzando questo oggetto, ma non ha ancora verificato la grammatica. La routine deve restituire `false` per continuare, `true` per indicare che la routine ha interpretato il comando od una parola del dizionario ('*parola*' o -'*parola*').

`in_to`
Vedere `d_to`.

`initial`
Per un *oggetto*: la descrizione visualizzata prima che venga raccolto.
Per una *stanza*: la descrizione visualizzata quando il giocatore entra nella stanza.
Il *valore* può essere una stringa od una routine che stampi una stringa.

`inside_description`
Per un oggetto `enterable`: la sua descrizione, visualizzata come parte della descrizione della stanza quando il giocatore è all'interno dell'oggetto.
Il *valore* può essere una stringa od una routine che stampi una stringa.

`invent`
Per un *oggetto*: il valore è una routine per la stampa dell'inventario dell'oggetto, viene chiamata due volte. Alla prima chiamata niente è stato ancora stampato; `inventory_stage` ha il valore 1, e la routine deve restituire `false` per continuare, o `true` per terminare l'azione e non produrre ulteriori output. Alla seconda chiamata sono stati stampati l'articolo indeterminativo dell'oggetto ed il suo nome sintetico (short name), ma non sono state stampate informazioni addizionali; il valore di `inventory_stage` è 2, e la routine deve restituire `false` per continuare, o `true` per terminare l'azione e non produrre ulteriori output.

`life` ♀
Per un oggetto `animate`: riceve le *azioni* da persona a persona (`Answer`, `Ask`, `Attack`, `Give`, `Kiss`, `Order`, `Show`, `Tell`, `ThrowAt` e `WakeOther`) per il quale questo è il `noun`. Il valore è una routine con una struttura simile a quella del costrutto `switch`, con una serie di casi per le azioni considerate (c'è anche un `default` opzionale). La routine deve restuire `false` per continuare, dicendo al giocatore quello che è accaduto, o `true` per interrompere l'azione e non produrre ulteriori output.

`list_together`
Per un oggetto: raggruppa gli oggetti correlati quando viene visualizzato l'elenco dell'inventario o degli oggetti presenti nella stanza.

Il *valore* può essere

- un *numero*: tutti gli oggetti che hanno questo valore vengono raggruppati;
- una *stringa*: tutti gli oggetti che hanno questo valore vengono raggruppati con il conteggio della stringa (ad es: cinque pesci);
- una routine che viene chiamata due volte. Alla prima chiamata niente è stato ancora stampato; `inventory_stage` ha il valore 1, e la routine deve restituire `false` per continuare, o `true` per terminare l'azione e non produrre ulteriori output. Alla seconda chiamata è stato stampato l'elenco; il valore di `inventory_stage` è 2 e non ci sono controlli sul valore di ritorno.

`n_to`
Vedere `d_to`.

`name` ♀
Definisce un elenco di parole separate da spazi che vengono aggiunte al dizionario di Inform. Ogni parola può essere fornita tra apici `'...'` o doppi apici `"..."`; in tutti gli altri casi solo le parole racchiuse tra apici vengono aggiunte al dizionario.
Per un *oggetto*: identifica quell'oggetto.
Per una *stanza*: visualizza "Non è importante ai fini del gioco."

`ne_to`
Vedere `d_to`.

`number`
Per un oggetto od una stanza: il *valore* è una variabile generale da usare liberamente nel programma. Un oggetto `player` [giocatore] deve fornire (ma non usare) questa variabile.

`nw_to`
Vedere *d_to*.

`orders`
Per un oggetto `animate` [animato] o `talkable` [a cui si può parlare]: il *valore* è una routine chiamata per gestire l'ordine del giocatore. La routine deve restituire `false` per continuare, oppure `true` per terminare l'azione e non produrre ulteriori output.

`out_to`
Vedere `d_to`.

`parse_name`
Per un oggetto: il *valore* è una routine chiamata per interpretare il nome dell'oggetto. La routine deve restituire 0 se il testo non ha senso, -1 per far continuare il parser con il suo lavoro, oppure un

numero positivo che indichi quante parole sono state interpretate.

`plural`
Per un oggetto: la sua forma plurale, quando si è un presenza di altri oggetti come questo. Il *`valore`* può essere una stringa od una routine che stampi una stringa.

`react_after`
Per un oggetto: individua le azioni che si svolgono nelle vicinanze, quelle che avvengono quando questo oggetto è in scope. Il *`valore`* è una routine che viene chiamata dopo che l'azione è accaduta, ma prima che il giocatore venga informato. Vedere `after`.

`react_before`
Per un oggetto: individua le azioni che si svolgono nelle vicinanze, quelle che avvengono quando questo oggetto è in scope. Il *`valore`* è una routine che viene chiamata prima che l'azione accada. Vedere `after`.

`s_to`
`se_to`
Vedere `d_to`.

`short_name`
Per un oggetto: un nome sintetico alternativo od aggiuntivo. Il *`valore`* può essere una stringa od una routine che stampi una stringa. La routine deve restituire `false` per continuare visualizzando il vero nome sintetico dell'oggetto (dalla testata della definizione dell'oggetto), oppure `true` per terminare l'azione e non produrre ulteriori output.

`short_name_indef`
Per un oggetto non inglese: il nome sintetico quando viene preceduto da un oggetto indefinito. Il *`valore`* può essere una stringa od una routine che stampi una stringa.

`sw_to`
Vedere `d_to`.

`time_left`
Per un oggetto timer: il *`valore`* è una variabile che contiene il numero di turni che mancano al timer associato a questo oggetto per arrivare a 0 ed invocare la proprietà `time_out` dello stesso (il timer viene attivato ed inizializzato con `StartTimer(`*`obj_id`*`)`).

`time_out`
Per un oggetto timer: il *`valore`* è una routine che viene eseguita quando il valore `time_left` dell'oggetto (inizializzato da `StartTimer(`*`obj_id`*`)` e non interrotto da `StopTimer(`*`obj_id`*`)`) raggiunge lo 0.

`u_to`
`w_to`
Vedere `d_to`.

`when_closed`
`when_opened`
Per un oggetto `container` [contenitore] o `door` [porta]: usato quando l'oggetto viene incluso nella descrizione estesa di una stanza. Il *`valore`* può essere una stringa od una routine che stampi una stringa.

`when_off`
`when_on`
Per un oggetto `switchable` [accendibile o attivabile]: usato quando l'oggetto viene incluso nella descrizione estesa di una stanza. Il valore può essere una stringa od una routine che stampi una stringa.

`with_key`
Per un oggetto `lockable` [chiudibile a chiave]: l'*`obj_id`* (solitamente una chiave) che serve per chiudere a chiave ed aprire l'oggetto, oppure `nothing` se nessuna chiave va bene.

Gli attributi degli oggetti

`absent`
Per un oggetto vagante (uno con la proprietà `found_in`, che può apparire in molte stanze): non è più lì.

`animate`
Per un oggetto: è una creatura vivente.

`clothing`
Per un oggetto: può essere indossato.

`concealed`
Per un oggetto: è presente, ma nascosto alla vista.

`container`
Per un oggetto: può contenere altri oggetti al suo interno (ma non sopra).

`door`
Per un oggetto: è una porta od un ponte tra due stanze.

`edible`
Per un oggetto: può essere mangiato.

`enterable`
Per un oggetto: ci si può entrare.

`female`
Per un oggetto `animate` [animato]: il sesso dell'oggetto è femminile. Tale attributo in italiano viene dato a tutti gli oggetti dal sostantivo femminile.

`general`
Per un oggetto od una stanza: un *`flag`* di uso generale.

`light`
Per un oggetto od una stanza: sorgente di luce.

`lockable`
Per un oggetto: può essere chiuso a chiave; vedere la proprietà `with_key`.

`locked`
Per un oggetto: non può essere aperto (è chiuso a chiave).

`male`
Per un oggetto `animate` [animato]: il sesso dell'oggetto è maschile, tale impostazione nelle librerie italiane è quella di default per tutti gli oggetti.

`moved`
Per un oggetto: il giocatore l'ha preso o lo sta prendendo.

`neuter`
Per un oggetto `animate` [animato]: l'oggetto non è né maschio né femmina.

`on`
Per un oggetto `switchable` [accendibile]: è acceso.

`open`
Per un oggetto `container` [contenitore] o `door` [porta]: è aperto.

`openable`
Per un oggetto `container` [contenitore] o `door` [porta]: può essere aperto.

`pluralname`
Per un oggetto: è plurale.

`proper`
Per un oggetto: l'oggetto ha un nome proprio, quindi non deve essere preceduto da "The" o "the", oppure "il", "lo", "la",...

`scenery`
Per un oggetto: non può essere preso; non viene visualizzato nella descrizione di una stanza.

`scored`
Per un oggetto: fa guadagnare `OBJECT_SCORE` punti quando viene preso per la prima volta. Per una stanza: fa guadagnare `ROOM_SCORE` punti quando vi si entra la prima volta.

`static`
Per un oggetto: non può essere preso.

`supporter`
Per un oggetto: vi si possono mettere sopra altri oggetti (ma non dentro).

`switchable`
Per un oggetto: può essere acceso o spento.

`talkable`
Per un oggetto: ci si può rivolgere con "oggetto, fai questo".

`transparent`
Per un oggetto `container` [contenitore]: gli oggetti che contiene sono visibili.

`visited`
Per una stanza: il giocatore la sta visitando o l'ha visitata.

`workflag`
`Flag` interno temporaneo, disponibile anche per il programma.

`worn`
Per un oggetto clothing [indossabile]: è indossato.

Punti d'ingresso opzionali

Queste routine, se le fornite, vengono chiamate quando indicato.

`AfterLife()`
Il giocatore è morto; impostare `deadflag`=0 lo fa resuscitare.

`AfterPrompt()`
Il prompt ">" è stato stampato.

`Amusing()`
Il giocatore ha vinto; è stata definita la costante `AMUSING_PROVIDED`.

`BeforeParsing()`
Il parser ha in ingresso del testo, ha impostato il buffer e le tabelle del parser ed ha inizializzato `wn` ad 1.

`ChooseObjects(`*`oggetto, flag`*`)`
Il parser ha trovato "ALL", "TUTTO" od una frase con un oggetto ambiguo (non è chiaro quale scegliere) ed ha deciso che l'oggetto deve essere escluso (se *`flag`* è impostato a 0), od incluso (se *`flag`* è impostato ad 1). La routine deve restituire 0 per accettare la decisione, 1 per forzare l'inclusione, 2 per forzare l'esclusione. Se il *`flag`* è 2 il parser è indeciso: la routine deve restituire un punteggio appropriato tra 0 e 9.

`DarkToDark()`
Il giocatore si è mosso da una stanza al buio ad un'altra.

`DeathMessage()`
Il giocatore è morto; `deadflag` vale 3 o superiore.

`GamePostRoutine()`
Chiamata dopo tutte le azioni.

`GamePreRoutine()`
Chiamata prima di tutte le azioni.

`Initialise()`
Obbligatoria; fare caso allo spelling britannico (la s al posto della z): chiamata ad ogni inizio di gioco. Deve impostare `location`; può restituire 2 per impedire la visualizzazione del banner.

`InScope()`
Chiamata durante il parsing (l'interpretazione del comando).

`LookRoutine()`
Chiamata alla fine di ogni descrizione `Look`, guarda.

`NewRoom()`
Chiamata quando la stanza cambia, prima di visualizzarne la descrizione.

`ParseNoun(`*`oggetto`*`)`
Chiamata per interpretare il nome dell'oggetto.

`ParseNumber(array_di_byte, lunghezza)`
Chiamata per interpretare un numero.

`PrintRank()`
Aggiunge informazioni alla visualizzazione del punteggio.

`PrintTaskName(`*`numero`*`)`
Stampa il nome del compito da eseguire.

`PrintVerb(`*`indirizzo`*`)`
Chiamata quando un verbo non usuale viene stampato.

`TimePasses()`
Chiamata dopo ogni turno.

`UnknownVerb()`
Chiamata quando un verbo sconosciuto viene incontrato.

Azioni del gruppo 1

Le azioni del gruppo 1 supportano i verbi 'meta'. Queste sono le azioni standard ed i verbi che le scatenano.

```
FullScore    "FULLSCORE", "FULL
             [SCORE]", "PUNTEGGIO
             COMPLETO", "PUNTI"

LMode1       "BRIEF", "NORMAL",
             "MODALITA NORMALE",
             "NORMALE"

LMode2       "LONG", "VERBOSE",
             "MODALITA
             LUNGA/COMPLETA", "LUNGO"

LMode3       "SHORT", "SUPERBRIEF",
             "MODALITA BREVE",
             "BREVE"

NotifyOff    "NOTIFY OFF", "NOTIFICA
             DISATTIVATA"

NotifyOn     "NOTIFY [ON]", "NOTIFICA
             ATTIVATA"

Objects      "OBJETCS", "OGGETTI"

Places       "PLACES", "LUOGHI",
             "POSTI"

Pronouns     "[PRO]NOUNS", "PRONOMI"

Quit         "DIE", "Q[UIT]",
             "USCIRE", "FINE",
             "BASTA"

Restart      "RESTART", "RICOMINCIA"

Restore      "RESTORE", "CARICA"

Save         "SAVE", "SALVA"

Score        "SCORE", "PUNTEGGIO"

ScriptOff    "[TRAN]SCRIPT OFF",
             "NOSCRIPT", "UNSCRIPT",
             "TRASCRIZIONE
             DISATTIVATA"

ScriptOn     "[TRAN]SCRIPT [ON]",
             "TRASCRIZIONE"

Verify       "VERIFY", "VERIFICA"

Version      "VERSION", "VERSIONE"
```

E gli strumenti di debug.

```
ActionsOff       "ACTIONS OFF"

ActionsOn        "ACTIONS [ON]"

ChangesOff       "CHANGES OFF"

ChangesOn        "CHANGES [ON]"

CommandsOff      "RECORDING OFF"

CommandsOn       "RECORDING [ON]"

CommandsRead     "REPLAY"

Gonear           "GONEAR"

Goto             "GOTO"

Predictable      "RANDOM"

RoutinesOff      "MESSAGES OFF",
                 "ROUTINES OFF"

RoutinesOn       "MESSAGES [ON]",
                 "ROUTINES [ON]"

Scope            "SCOPE"

Showobj          "SHOWOBJ"

Showverb         "SHOWVERB"

TimersOff        "DAEMONS OFF",
                 "TIMERS OFF"

TimersOn         "DAEMONS [ON]",
                 "TIMERS [ON]"

TraceLevel       "TRACE numero"
```

```
TraceOff        "TRACE OFF"
TraceOn         "TRACE [ON]"
XAbstract       "ABSTRACT"
XPurloin        "PURLOIN"
XTree           "TREE"
```

Azioni del gruppo 2

Le azioni del gruppo 2 solitamente funzionano se vengono fornite le corrette circostanze.

```
Close       "CHIUDI", "COPRI"
Disrobe     "RIMUOVI", "TOGLI"
Drop        "LASCIA", "LANCIA",
            "ABBANDONA", "POSA",
            "METTI GIU"
Eat         "MANGIA"
Empty       "SVUOTA"
EmptyT      "SVUOTA
            SU|SUL|SULLO|SULL'|SULLA
            |SUI|SUGLI|SULLE|SOPRA"
Enter       "STAI SU|SUL|SULLO...",
            "STA SU|SUL|SULLO...",
            "VAI", "CAMMINA",
            "CORRI", "VA", "VAI
            IN|NEL|NELLO|NELL'|NELLA
            |NEI|NEGLI|NELLE|DENTRO|
            ATTRAVERSO|A|AD|ALL'|ALL
            O|ALLA|AL|AGLI|AI|ALLE",
            "CAMMINA
            IN|NEL|NELLO...", "CORRI
            IN|NEL|NELLO...", "VA
            IN|NEL|NELLO...",
            "ENTRA", "ATTRAVERSA",
            "VISITA", "ENTRA
            IN|NEL|NELLO...",
            "ATTRAVERSA
            IN|NEL|NELLO...",
            "SIEDI", "SIEDITI",
            "SDRAIATI", "SIEDI
            SU|SUL|SULLO|SULL'|SULLA
            |SUI|SUGLI...|IN|NEL|NEL
            LO...", "SIEDITI
            SU|SUL...", "SDRAIATI
            SU|SUL..."
Examine     "G[UARDA]", "VEDI", "L",
            "ESAMINA", "X",
            "DESCRIVI", "CONTROLLA",
            "LEGGI"
Exit        "LASCIA", "ABBANDONA",
            "VAI FUORI", "CAMMINA
            FUORI", "CORRI FUORI",
            "VA FUORI", "SCENDI",
            "SCENDI
            DA|DAL|DALLO|DALLA|DALL'
            |DAI|DAGLI|DALLE",
            "FUORI", "ESCI", "FUORI
            DA|DAL|DALLO...", "ESCI
            DA|DAL|DALLO..."
Give        "DAI
            A|AD|ALL'|ALLO|ALLA|AL|A
            GLI|AI|ALLE", "PAGA
            A|AD...", "OFFRI
            A|AD...", "DA A|AD..."
Go          "LASCIA", "ABBANDONA",
            "VAI A|AD|VERSO",
            "CAMMINA A|AD|VERSO",
            "CORRI A|AD|VERSO", "VA
            A|AD|VERSO"
GoIn        "VAI DENTRO", "CORRI
            DENTRO", "VA DENTRO",
            "ENTRA", "ATTRAVERSA",
            "VISITA", "ENTRA
            DENTRO", "IN", "DENTRO"
Insert      "METTI
            DENTRO|IN|NEL|NELLO|NELL
            '|NELLA|NEGLI|NELLE|NEI"
            , "INSERISCI
            DENTRO|IN|NELLO...",
            "LASCIA
            DENTRO|IN|NELLO...",
            "ABBANDONA
            DENTRO|IN|NELLO...",
            "POSA
            DENTRO|IN|NELLO..."
Inv         "FAI INVENTARIO",
            "INVENTARIO", "INV", "I"
InvTall     "INVENTARIO ESTESO",
            "INV ESTESO", "I ESTESO"
InvWide     "INVENTARIO COMPLETO",
            "INV COMPLETO", "I
            COMPLETO"
Lock        "CHIUDI CON|COL|A",
            "SERRA CON|COL", "BLOCCA
            CON|COL"
Look        "GUARDA", "G", "VEDI",
            "L"
Open        "APRI", "SCOPRI"
PutOn       "METTI
            SU|SUL|SULLO|SULL'|SULLA
            |SUI|SUGLI|SULLE|SOPRA",
            "METTITI
            SU|SUL|SULLO...",
            "LASCIA
            SU|SUL|SULLO...",
            "LANCIA
            SU|SUL|SULLO...",
            "ABBANDONA
            SU|SUL|SULLO...", "POSA
            SU|SUL|SULLO..."
Remove      "PRENDI
            DA|DAL|DALLO|DALLA|DALL'
            |DAGLI|DALLE|DAI",
            "TRASPORTA
            DA|DAL|DALLO...",
            "AFFERRA
            DA|DAL|DALLO...",
            "RACCOGLI
            DA|DAL|DALLO...",
            "RIMUOVI
            DA|DAL|DALLO...", "TOGLI
            DA|DAL|DALLO..."
```

```
Search      "G[GUARDA]
            DENTRO|IN|NEL|NELLO|NELL
            '|NELLA|NEGLI|NELLE|NEI|
            ATTRAVERSO|SU|SUL|SULLO|
            SULL'|SULLA|SUI|SUGLI|SU
            LLE|SOPRA", "VEDI
            DENTRO|IN|NEL...", "L
            DENTRO|IN|NEL...",
            "CERCA
            [DENTRO|IN|NEL|NELLO|NEL
            L'|NELLA|NEGLI|NELLE|NEI
            ]", "TROVA
            [DENTRO|IN|NEL...]",
            "RICERCA
            [DENTRO|IN|NEL...]"

Show        "MOSTRA
            A|AD|ALL'|ALLO|ALLA|AL|A
            GLI|AI|ALLE", "PRESENTA
            A|AD|ALL'...", "FAI
            VEDERE A|AD|ALL'..."

SwitchOff   "GIRA A OFF|SU OFF",
            "RUOTA A OFF|SU OFF",
            "DISATTIVA", "SPEGNI"

SwitchOn    "GIRA A ON|SU ON",
            "RUOTA A ON|SU ON",
            "ATTIVA", "ACCENDI"

Take        "PRENDI", "TRASPORTA",
            "AFFERRA", "RACCOGLI",
            "RIMUOVI", "TOGLI",
            "PELA"

Transfer    "TRASFERISCI
            SU|SUL|SULLO|SULL'|SULLA
            |SUI|SUGLI|SULLE|SOPRA|D
            ENTRO|IN|NEL|NELLO|NELL'
            |NELLA|NEGLI|NELLE|NEI",
            "SPOSTA SU|SUL|SULLO..."

Unlock      "APRI CON|COL|A",
            "SCOPRI CON|COL|A",
            "SCASSINA CON|COL|A",
            "SBLOCCA CON|COL|A"

VagueGo     "VAI", "CAMMINA",
            "CORRI", "VA"

Wear        "INDOSSA", "METTI",
            "METTITI"
```

Le azioni del gruppo 3

Le azioni del gruppo 3 sono per impostazione predefinita dei modelli che visualizzano un messaggio e si fermano alla fase "before" [prima] (quindi non c'è una fase "after" [dopo]).

```
Answer      "RISPONDI
            A|AD|ALL'|ALLO|ALLA|AL|A
            GLI|AI|ALLE", "DÌ
            A|AD|ALL'...", "GRIDA
            A|AD|ALL'...", "DI
            A|AD|ALL'...", "DI'
            A|AD|ALL'...", "DICI
            A|AD|ALL'..."

Ask         "CHIEDI A|AD|ALL'...
            CIRCA|SU|SUL|SUI|SULLO|S
            ULL'|SULLA|SUGLI|SULLE|D
            I|DELLO|DELLA|DELL'|DEI|
            DEGLI|DELLE", "DOMANDA
            A|AD|ALL'...
            CIRCA|SU|SUL..."

AskFor      "CHIEDI A|AD|ALL'...",
            "DOMANDA A|AD|ALL'..."

Attack      "ATTACCA [CON|COL]",
            "ROMPI [CON|COL]",
            "COLPISCI [CON|COL]",
            "COMBATTI [CON|COL]",
            "UCCIDI [CON|COL]",
            "TORTURA [CON|COL]",
            "LOTTA [CON|COL]",
            "SFONDA [CON|COL]",
            "AMMAZZA [CON|COL]"

Blow        "SOFFIA
            [DENTRO|IN|NEL|NELLO|NEL
            L'|NELLA|NEGLI|NELLE|NEI
            ]"

Burn        "BRUCIA [CON|COL]",
            "INCENDIA [CON|COL]"

Buy         "COMPRA", "ACQUISTA"

Climb       "VAI
            SU|SUL|SULLO|SULL'|SULLA
            |SUI|SUGLI|SULLE|SOPRA",
            "CAMMINA
            SU|SUL|SULLO...", "CORRI
            SU|SUL|SULLO...", "VA
            SU|SUL|SULLO...", "SCALA
            [SU|SUL|SULLO...]",
            "SALI
            [SU|SUL|SULLO...]",
            "ARRAMPICA
            [SU|SUL|SULLO...]",
            "ARRAMPICATI
            [SU|SUL|SULLO...]"

Consult     "CONSULTA
            CIRCA|SU|SUL|SULLO|SULL'
            |SULLA|SUI|SUGLI|SULLE|S
            OPRA", "LEGGI
            DENTRO|IN|NEL|NELLO|NELL
            '|NELLA|NEGLI|NELLE|NEI"

Cut         "TAGLIA [CON|COL]",
            "AFFETTA [CON|COL]",
            "SFRONDA [CON|COL]",
            "SFOLTISCI [CON|COL]",
            "SPACCA [CON|COL]",
            "STRAPPA [CON|COL]"

Dig         "SCAVA [CON|COL]"

Drink       "BEVI", "TRANGUGIA",
            "SORSEGGIA"

Fill        "RIEMPI", "COLMA"

Jump        "SALTA"

JumpOver    "SALTA
            SU|SUL|SULLO|SULL'|SULLA
            |SUI|SUGLI|SULLE|SOPRA"

Kiss        "BACIA", "ABBRACCIA"

Listen      "SENTI", "ASCOLTA"
```

LookUnder	"G[UARDA] SOTTO", "VEDI SOTTO", "L SOTTO"
Mild	Diverse imprecazioni minori
No	"NO"
Pray	"PREGA"
Pull	"TIRA", "TRASCINA"
Push	"TRASFERISCI", "SPOSTA", "PREMI", "MUOVI", "SPINGI"
PushDir	"PREMI A|AD|ALL'|ALLO|ALLA|AL|AGLI|AI|ALLE", "MUOVI A|AD|ALL'..."
Rub	"PULISCI", "STROFINA", "SPOLVERA", "RIPULISCI", "LUCIDA", "LUSTRA"
Set	"IMPOSTA"
SetTo	"IMPOSTA TO"
Sing	"CANTA"
Sleep	"DORMI", "SONNECCHIA"
Smell	"ANNUSA", "ODORA"
Sorry	"SPIACENTE", "SCUSA", "CHIEDI SCUSA|SCUSE", "DOMANDA S CUSA|SCUSE"
Squeeze	"SCHIACCIA", "SPREMI", "SPIACCICA"
Strong	Diverse imprecazioni forti.
Swim	"NUOTA"
Swing	"SCUOTI"
Taste	"ASSAGGIA", "ASSAPORA", "GUSTA"
Tell	"PARLA A|AD|ALL'|ALLO|ALLA|AL|AGLI|AI|ALLE CIRCA|SU|SUL|SULLO|SULL'|SULLA|SUGLI|SUI|SULLE|DI|DELLO|DELLA|DELL'|DEI|DEGLI|DELLE", "PARLA CON|COL [CIRCA|SU|SUL...]"
Think	"PENSA", "MEDITA", "RIFLETTI"
ThrowAt	"LANCIA A|AD|ALL'|ALLO|ALLA|AL|AGLI|AI|ALLE|SU|SUL|SULLO|SULL'|SULLA|SUI|SUGLI|SULLE|SOPRA|CONTRO|DENTRO|IN|NEL|NELLO|NELL'|NELLA|NEGLI|NELLE|NEI"
Tie	"LEGA [A|AD|ALL'|ALLO|ALLA|AL|AGLI|AI|ALLE] [CON|COL]", "FISSA [A|AD|ALL'...] [CON|COL]", "CONGIUNGI [A|AD|ALL'...] [CON|COL]", "UNISCI [A|AD|ALL'...] [CON|COL]", "ALLACCIA [A|AD|ALL'...] [CON|COL]", "ANNODA [A|AD|ALL'...] [CON|COL]"
Touch	"TOCCA", "ACCAREZZA", "PALPA"
Turn	"GIRA", "RUOTA"
Wait	"ASPETTA", "ATTENDI", "Z"
Wake	"SVEGLIA", "SVEGLIATI", "RISVEGLIA", "RISVEGLIATI"
WakeOther	"SVEGLIA", "SVEGLIATI", "RISVEGLIA", "RISVEGLIATI"
Wave	"AGITA", "AGITATI"
WaveHands	"SALUTA"
Yes	"SI", "SÌ"

Azioni finte (fake action)

Le azioni finte gestiscono alcuni casi speciali, oppure rappresentano azioni "vere" dal punto di vista del secondo oggetto.

LetGo	Generata da `Remove`
ListMiscellany	Visualizza un insieme di messaggi di inventario.
Miscellany	Visualizza un insieme di messaggi di utilità.
NotUnderstood	Generato quando il parser non riesce ad interpretare alcuni ordini (`order`).
Order	Riceve le cose non gestite da `orders`.
PluralFound	Dice al parser che `parse_name()` ha indentificato un oggetto plurale.
Prompt	Visualizza il prompt, di solito ">".
Receive	Generata da `Insert` e `PutOn`.
TheSame	Generata quando il parser non è in grado di distinguere due oggetti.
ThrownAt	Generata da `ThrowAt`.

Appendice G: Glossario

Durante la nostra avventura abbiamo incontrato alcuni termini che hanno un significato particolare nell'ambito del sistema di sviluppo per avventure testuali Inform: quelle che seguono sono delle brevi definizioni di molte di queste parole e frasi.

action – vedi **azione**

albero degli oggetti – una gerarchia che definisce le relazioni tra oggetti nei termini di chi contiene chi. Ogni **oggetto** può essere sia contenuto all'interno di un altro oggetto (il genitore) o *non* contenuto; gli oggetti come le stanze che non si trovano in un altro oggetto hanno come genitore la **costante** `nothing` (0). Un oggetto contenuto in un altro oggetto è un figlio. Ad esempio per una scarpa all'interno della scatola: la scatola è il genitore della scarpa e la scarpa è un figlio della scatola. Adesso supponiamo che la scatola si trovi nell'armadio. La scatola è un figlio dell'armadio, ma la scarpa è ancora un figlio della scatola, non dell'armadio. In un gioco normale l'albero degli oggetti subirà numerose trasformazioni in seguito alle attività del giocatore.

alpha-testing – (anche alfa test) il test che viene fatto dal creatore del gioco, in un vano tentativo di assicurarsi che faccia tutto ciò che dovrebbe e niente di quello che non dovrebbe. Vedere anche **beta-testing**.

analizzatore sintattico – vedi **parser**

argomento – un parametro fornito in una chiamata ad una **routine**, che diventa il valore di una delle variabili locali definite nella routine. Ad esempio l'argomento è 8 nella chiamata a MiaRoutine(8). La definizione della routine include la variabile che conterrà l'argomento, nel nostro caso x: [MiaRoutine x; ...];

argument – vedi **argomento**

assegnazione – un'istruzione che imposta o cambia il valore di una **variabile**. Ce ne sono tre in Inform: `=` (imposta uguale a), `++` (aggiungi uno al valore corrente), `--` (sottrai uno dal valore corrente).

attribute – vedi **attributo**

attributo – **flag** con nome che possono essere definiti per un oggetto dopo la parola chiave `has` [ha]. Un attributo può essere presente (on/acceso) o non presente (off/spento). Lo scrittore può verificare in ogni parte del programma se [if] un oggetto ha [`has`] un determinato attributo, dare [`give`] un attributo ad un oggetto o levarglielo a seconda delle necessità. Ad esempio l'attributo `container` [contenitore] indica che l'oggetto è in grado di contenere altri oggetti al suo interno.

avatar – vedi **giocatore**

azione – l'effetto generato dall'input del giocatore, trattato solitamente dal **parser** od anche dal codice dello scrittore di avventure. Si riferisce ad un singolo compito che deve essere processato da

Inform, come in LASCIA IL BRICCO, ed è memorizzato in quattro numeri: uno per l'azione ed uno per l'oggetto `actor` [attore] che la esegue (il giocatore od un NPC), uno per il `noun` (il complemento oggetto), chiamato anche oggetto diretto se presente, ed un quarto numero per il secondo `noun`, se esiste, come ad esempio la "TEIERA" in TIRA IL BRICCO ALLA TEIERA.

banner – le informazioni riguardanti il gioco che vengono mostrate all'inizio della partita.

beta-testing – (anche beta test) il test che viene effettuato da una banda di fidati volontari, prima del rilascio al pubblico, durante il quale la rozza inadeguatezza dell'**alpha-testing** dello scrittore viene, dolorosamente, a galla.

binary file – vedi **file binario**

blocco istruzioni – un gruppo di **istruzioni** che si trova tra parentesi graffe `{...}` e che vengono trattate come un'unità singola, come se fossero un'unica istruzione. Solitamente compaiono nei cicli e nelle condizioni.

bold type – vedi **grassetto**

child – (figlio) vedi **albero degli oggetti**

class – uno speciale modello di **oggetto** dal quale altri oggetti possono ereditare **proprietà** ed **attibuti**. Il modello deve cominciare con la parola chiave `Class` e deve avere un identificatore interno. Gli oggetti che desiderano ereditare da una classe di solito cominciano con l'ID interno della classe al posto della parola Object, ma possono anche definire un segmento `class` seguito dall'ID interno della classe. Lo scrittore di avventure può verificare se un oggetto appartiene ad (se è un membro di) una classe.

classe – vedi **class**

code block – vedi **blocco istruzioni**

codice sorgente – vedi **file sorgente**

commento – del testo che comincia con un punto esclamativo `!` e che viene ignorato dal **compilatore** quando legge il **file sorgente**; usato per migliorare l'aspetto del file od aggiungere note di spiegazione.

compilatore – un programma che legge il codice sorgente scritto dal programmatore e lo trasforma in uno **story file** che può essere giocato con un **interprete** Z-Machine.

compile-time – vedi **tempo di compilazione**

compiler – vedi **compilatore**

corsivo – usato per enfatizzare, e come un indicatore che rappresenta un valore che dovete fornire.

costante – un particolare valore che viene definito a **tempo di compilazione** che resta sempre lo stesso e non può essere cambiato mentre il gioco è in esecuzione. Esempi comuni comprendono i numeri, le stringhe e gli ID interni degli oggetti, ognuno dei quali può essere scritto esplicitamente oppure impostato come il valore di una

costante con nome (con la parola chiave `Constant`).

default – il valore predefinito di un determinato elemento, usato in mancanza di un'assegnazione esplicita.

designer – vedi **scrittore**

dictionary word – vedi **parola del dizionario**

dictionary - vedi **dizionario**

directive – vedi **direttiva**

direttiva – una riga di codice Inform che chiede al **compilatore** di fare qualcosa in quel punto, a **tempo di compilazione**; un tipico esempio è quello di voler includere (`Include`) il contenuto di un altro file, oppure di riservare dello spazio nello **story file** per poterci memorizzare il valore di una variabile. Da non confondere con un'**istruzione**, che chiede al **compilatore** di creare un comando che verrà eseguito dall'interprete a **tempo di esecuzione**: un tipico esempio è quello di far visualizzare un testo o di cambiare il valore contenuto nello spazio di immagazzinamento di una variabile.

dizionario – l'insieme di tutte le parole in input che il gioco "capisce".

editor – un programma che si usa per creare e modificare i **file di testo**.

entry point - vedi **punto d'ingresso**

ereditarietà - il processo tramite il quale un **oggetto** appartenente ad una **classe** acquisisce le **proprietà** e gli **attributi** di quella classe. L'ereditarietà è automatica, lo scrittore di avventure deve solo creare le definizioni delle classi, seguite dagli oggetti che hanno quelle classi.

false – uno stato logico che è l'opposto di **true**, rappresentato dal valore 0.

falso – vedi **false**

figlio – vedi **albero degli oggetti**

file ASCII – vedi **file di testo**

file binario – un file del computer che contiene dati binari (tutti 0 e 1) che viene creato da un programma e che solo un programma può capire.

file della libreria – i file che contengono il codice sorgente della **libreria**. Sostanzialmente sono tre se scrivete un gioco in Inglese (sebbene questi tre includano anche altri file): `parser.h`, `verblib.h` e `grammar.h` e devono essere inlcusi in ogni gioco Inform. In Italiano i file da includere sono quattro: `parser.h`, `verblib.h`, `replace.h` e `italiang.h`.

file sorgente – un file di testo che contiene il vostro gioco definito usando il linguaggio Inform.

file testo – un file del computer che contiene parole e frasi che una persona può leggere.

file Z-code – vedi **story file**

flag – una variabile che può contenere solo due possibili valori.

function – vedi **routine**

funzione – vedi **routine**

genitore – vedi **albero degli oggetti**

giocatore – l'utente finale del gioco, di solito è una persona piena di opinioni radicali riguardo le vostre capacità di scrittore. Vedi anche **player**.

grassetto – usato per evidenziare un termine spiegato in questo glossario.

ICGAT – il newsgroup `it.comp.giochi.avventure.testuali` per la comunità italiana di scrittori e giocatori di IF.

Infit – è la traduzione italiana della libreria originale inglese. Permette la creazione di giochi scritti in lingua italiana. E' composta di tre file `italian.h` che sostituisce l'originale `english.h`, `italianG.h` che sostituisce l'originale `grammar.h` e `replace.h` che sostituisce alcune funzioni contenute nei file di libreria `parser.h` e `verblib.h`. L'attuale versione è la 2.5

inheritance – vedi **ereditarietà**

interprete – un programma che legge lo **story file** di un gioco e consente alle persone di giocarci. Gli interpreti sono specifici per ogni piattaforma (cioè sono programmi differenti per ogni sistema operativo) consentendo quindi allo story file di essere universale ed indipendente dalla piattaforma.

istruzione – un comando che deve essere eseguito a **tempo di esecuzione**. Vedere anche **direttiva**.

library – vedi **libreria**

libreria – un gruppo di file di testo, facenti parte del sistema Inform, che include il **parser**, le definizioni per il **model world**, i file delle lingue, le definizioni della grammatica ed un insieme di risposte predefinite e comportamenti standard alle azioni del giocatore. La libreria fa delle chiamate molto frequenti al gioco per vedere se il programmatore vuole evitare questi comportamenti predefiniti.

locazione – vedi **stanza**

model world – l'ambiente immaginario nel quale il personaggio del giocatore vive.

modello del mondo – vedi **model world**

modo Debug – un'opzione che fa in modo che il **compilatore** includa del codice aggiuntivo nello **story file** così da rendere più semplice per il programmatore capire cosa sta succedendo mentre un gioco viene testato prima del rilascio. Vedere anche **modo Strict**.

modo Strict – un'opzione che fa in modo che il **compilatore** includa del codice aggiuntivo nello **story file** così che sia più semplice individuare determinati errori di progettazione mentre si sta giocando. Questo modo attiva automaticamente il **modo Debug**.

newline – il carattere di controllo ASCII usato per indicare la fine di una riga di testo.

newsgroup – conosciuti anche come forum e gruppi di discussione.

NPC – (non-player character) vedi **PNG**

object tree – vedi **albero degli oggetti**

object – vedi **oggetto**

oggetto - un gruppo di **routine** e **variabili** unite insieme in un'unità coerente. Gli oggetti rappresentano le cose che creano il **model world** (una torcia, una macchina, un raggio di luce, ecc.), è un qualcosa che organizza il codice del programmatore in pezzi distinti, facili da trattare. Ogni oggetto ha due parti: l'intestazione, che comprende l'ID interno, il nome esterno ed i suoi parenti definiti (tutti i campi sono opzionali), ed il corpo, che comprende le variabili di proprietà ed i flag degli attributi specifici di quell'oggetto (se ce ne sono).

parent – (genitore) vedi **albero degli oggetti**

parola del dizionario – una parola scritta tra apici `'...'` nel **file sorgente**, di solito (ma non solo) è uno dei valori assegnati alla proprietà name di un oggetto. Tutte quelle parole vengono memorizzate nel **dizionario**, che viene interrogato dal **parser** quando cerca di capire il comando del giocatore. Solo i primi nove caratteri vengono presi in considerazione (quindi `'cardiogramma'` e `'cardiografo'` vengono trattati come se fossero la stessa parola). Si usa `'monete//p'` per impostare "monete" come plurale, riferendosi a tutti gli oggetti moneta che sono presenti. Si usa `'t//'` per inserire la parola di un solo carattere "t" nel dizionario (`'t'`, invece, è una costante che rappresenta un valore carattere).

parser – il parser è quella parte della **libreria** che si prende cura di analizzare l'input del giocatore e cerca di capire cosa vuole, dividendolo in parole (verbi, nomi), e confrontando il tutto con le parole che sono memorizzate nel **dizionario** del gioco e con le azioni definite nella grammatica. Se l'input del giocatore ha un qualche significato il parser attiva l'**azione** risultante, altrimenti si lamenta dicendo che non capisce cosa è stato scritto.

PC – 1. un personal computer; 2. il personaggio del giocatore (vedi **giocatore**)

PG – il personaggio del giocatore. Vedi **giocatore**

player – la variabile che si riferisce all'**oggetto** (conosciuto anche come "avatar") che rappresenta il giocatore all'interno del **model world**. Vedi anche **giocatore**

PNG – personaggio non giocatore/personaggio non giocante: un qualunque personaggio che non è il protagonista. Spazia dai "nemici" agli amanti, da un gerbillo domestico fino ad un pedone incontrato per caso.

print rule – la regola da applicare in un'istruzione `print` o `print_ret` per controllare la maniera con la quale un determinato dato deve essere visualizzato. Ad esempio: `print (The) noun, "@`e qui."` dice al gioco di usare l'articolo definito con la prima lettera maiuscola per `noun`. La **libreria** definisce un insieme di *print rule*, e

lo scrittore può aggiungere le proprie.

properties – vedi **proprietà**

property – vedi **proprietà**

proprietà – variabili associate ad un singolo **oggetto** del quale fanno parte. Vengono definite nel corpo dell'oggetto dopo la parola chiave `with` ed hanno un nome ed un valore. Quest'ultimo (che per impostazione predefinita vale 0) può essere un numero, una stringa `"..."`, una parola del dizionario `'...'` od una routine incorporata `[;...];` può anche essere un elenco di questi separati da spazi. Il valore della proprietà di un oggetto può essere letto e modificato in ogni punto del gioco. Anche il fatto che un oggetto disponga di una determinata proprietà può essere verificato.

punto d'ingresso – una tra un elenco predefinito di routine opzionali che, se fornita, verrà chiamata dalla libreria per produrre dell'output aggiuntivo o per restituire un valore che cambi il comportamento standard della libreria.

RAIF – il newsgroup `rec.arts.int-fiction` per gli scrittori di IF.

RGIF – il newsgroup `rec.games.int-fiction` per i giocatori di IF.

room – vedi **stanza**

routine della libreria – una tra un insieme di routine incluse come parte della **libreria** che il programmatore può chiamare per eseguire alcuni utili compiti comuni.

routine embedded – una routine che viene definita nel corpo di un oggetto, come valore di una delle sue **proprietà**. Diversamente da una **routine standalone**, una routine embedded non ha un nome e restituisce `false` quando l'esecuzione raggiunge la parentesi quadra chiusa `]` che ne indica il termine.

routine incorporata - vedi **routine embedded**

routine indipendente – vedi **routine standalone**

routine standalone – una routine che non fa parte di un oggetto. Diversamente da una **routine embedded** deve avere un proprio nome e restituisce `true` quando l'esecuzione raggiunge la parentesi quadra chiusa `]` che ne indica il termine.

routine – in termini generali una routine è un programma per computer che effettua alcuni calcoli specifici, seguendo un insieme ordinato di istruzioni: è la sola unità di codice coerente ed eseguibile conosciuta da Inform. In pratica una routine è un insieme di **istruzioni** che sono scritte tra gli identificatori `[...]`. Quando una routine viene "chiamata", possibilmente con degli argomenti (valori specifici per le sue variabili definite, se ce ne sono), l'interprete esegue le sue istruzioni in sequenza. Se l'interprete incontra una istruzione `return`, o raggiunge la `]` alla fine della routine, termina immediatamente l'esecuzione delle

istruzioni della routine e riprende l'esecuzione dall'istruzione che ha chiamato la routine. Ogni routine restituisce un valore, che può essere fornito dall'istruzione `return` oppure è causato dalla `]` alla fine della routine. Vedere anche **routine embedded** e **routine standalone**.

run-time – vedi **tempo di esecuzione**

scrittore [di avventure] – la persona che usa Inform per creare un'avventura testuale: in altre parole, gentile lettore, tu.

source file – vedi **file sorgente**

stanza – un **oggetto** che definisce un'unità geografica nella quale il **model world** è diviso. Le stanze non hanno oggetti genitore (più precisamente, il loro oggetto genitore è `nothing`) e rappresentano i posti nei quali il personaggio del giocatore si trova in un determinato momento: il personaggio del giocatore non può trovarsi in più di una stanza alla volta. Fate caso al fatto che "stanza" non implica necessariamente "all'interno". Un prato, una spiagga, la cima di un albero od anche galleggiare nello spazio: questi sono tutti possibili oggetti stanza.

statement – vedi **istruzione**

story file – un file binario che è l'output del **compilatore** e che può essere giocato attraverso l'uso di un **interprete** (viene anche chiamato file Z-code o game file). Il formato degli story file è standard ed indipendente dalla piattaforma.

stringa – una parte di testo racchiusa tra virgolette `"..."` per essere mostrata al giocatore a **tempo di esecuzione**.

switch – 1. una parola chiave opzionale od un simbolo che attivano funzionalità speciali del compilatore. 2. un'istruzione che sceglie tra diversi percorsi di esecuzione in base al valore di un'espressione.

tempo di compilazione – si intende per "tempo di compilazione" il tempo durante il quale il **compilatore** lavora per produrre lo **story file** (l'espressione usata di solito è "a tempo di compilazione"). Vedi anche **tempo di esecuzione**.

tempo di esecuzione – il periodo di tempo nel quale l'**interprete** sta eseguendo uno **story file** (cioè qualcuno sta giocando con il gioco). Vedere anche **tempo di compilazione**.

true – uno stato logico che è l'opposto di **false**, rappresentato da qualunque valore diverso da 0 (solitamente 1).

variabile globale – una variabile che non è specifica di una routine o di un oggetto e che può essere usata da ogni routine od oggetto nel gioco.

variabile locale – una variabile che fa parte di una determinata **routine**: il suo valore non è accessibile alle altre routine del gioco. Il valore di una variabile locale *non* viene conservato tra una chiamata alla routine e le successive.

variabile – un valore con nome che può cambiare a **tempo di esecuzione**. Deve essere dichiarata

prima del suo uso, sia come variabile globale (disponibile ad ogni routine del gioco) che come variabile locale (parte di una specifica routine ed utilizzabile solo da quella routine). Le variabili hanno un nome ed un valore; è il valore che cambia, non il nome. Le **proprietà** degli oggetti si comportano come variabili.

vero – vedi **true**

Z-Machine – una macchina virtuale (un computer immaginario simulato dall'**interprete**) nella quale vengono eseguiti gli **story file**. La Z sta per "Zork", il primo titolo della Infocom.

Indice

Simboli

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

Z