1 Comenzando con InformATE!

Resumen. *En este artículo se verá, desde el punto de vista del programador de C++, cómo empezar a trabajar con Inform. Al final del mismo se obtendrá una mini-aventura completa que puede ser utilizada como ejemplo para futuros trabajos.*

1. Introducción

En este pequeño artículo, veremos como comenzar a utilizar Inform [1] de Graham Nelson, con las librerías traducidas para español de Zak MacKracken [2]. Este artículo no debe ser por tanto considerado como una referencia exhaustiva de InformatE!, ya que para eso ya está DocumentatE! [3] (por otra parte, hacer esto excedería la extensión de un artículo).

Inform es un compilador, que acepta código fuente Inform y genera código nativo, no para chips Intel, que es lo habitual en un compilador (de C/C++, por ejemplo), sino para una máquina especial conocida como 'Z'. La versión ahora mismo más utilizada de esta máquina es la 5. Los ficheros generados por inform tienen la extensión '.z5'.

La pregunta que surge es tratar de imaginase cuál puede ser la motivación de crear una máquina especial, llamada 'Z'. La respuesta es crear una máquina (es decir, un ordenador, para entendernos) con una orientación especial para soportar la ejecución de aventuras conversacionales. Evidentemente, como el lector habrá adivinado ya, nadie va a fabricar especialmente un ordenador sólo para jugar aventuras conversacionales, así que es necesario crear un intérprete que convierta una plataforma cualquiera (por ejemplo, un intel con SO (Sistema Operativo) Win32, o el mismo intel corriendo Linux) en una máquina Z [5]. Ésta parte de la tarea es la que tiene encargada el frotz [4], que precisamente es ese intérprete. Por tanto, podemos imaginar que hay un frotz para la mayoría de las plataformas.

Existe la posibilidad de crear aventuras con ingredientes gráficos. La máquina frotz no soporta gráficos, por lo menos en la versión 5. La versión 6 sí los soporta, pero necesitarías un compilador adecuado. Existe también el proyecto Iznoguz, que es un proyecto que puedes bajarte ya y que soporta gráficos con Inform [6][7].

Las ventaja de cara al programador acerca de utilizar la máquina Z o no, es una, muy clara: el fichero generado por Inform, el que tiene extensión '.z5' puede ser ejecutado por cualquier persona que disponga de un intérprete de máquina Z, tenga un PC, un Mac, un Sparc, con operativo Win95, Unix o Linux. El esfuerzo de traducción a distintas plataformas ya se hizo con frotz, y tú, para tu aventura, no tienes que hacer nada extra para que varias personas con computadoras con SO's distintos puedan ejecutarla.

Además, es mucho más fácil traducir la máquina Z (que tiene una especificación muy concreta) a varias plataformas, que traducir cada una de las posibles aventuras. Esto es además posible gracias a que los intérpretes suelen estar construidos en C, que resulta ser un lenguaje muy portable.

2. Bajándote el Software

Por tanto, lo necesario para comenzar con Inform [2] es:

- Dependiente de la plataforma (van por separado):
  - El compilador Inform.
  - El intérprete de la máquina Z (frotz).
- Independiente de la plataforma:
  - Librería InformatE! para aventuras en español.
  - La documentación, DocumentatE!

Tendrás que bajártelo en tres paquetes, es decir:

- Desarrollo en Inform (para cada plataforma, que incluye la librería InformatE!).
- DocumentatE! (el cursillo de programación).
- Intérprete frotz (para cada platforma).

La instalación es extremadamente sencilla. Basta colocar los paquetes en un directorio, y descomprimirlos allí [8].

Para comprobar que todo está bien, y siguiendo la tradición, aquí está el programa Hola, Mundo:

Tecléese lo siguiente en un editor de textos, y guárdalo como hola.inf:

```
! Programa de ejemplo "Hola, mundo"
[ Main;
  print "Hola, mundo^";
];
```

Ahora, lo compilamos:

```
inform hola.inf
DJGPP Inform 6.15 (22nd March 1998)
```

Y finalmente, lo ejecutamos:

```
frotz hola.z5
Hola, mundo !
```

Si lo anterior aparece en pantalla, InformATE! Está instalado satisfactoriamente y es posible comenzar a desarrollar aventuras. En este ejemplo, no se ha utilizado en realidad la librería InformATE!, pero como te la bajas del paquete de la página de Zak, es seguro que la tienes.

Si trabajas en Windows, puedes automatizar la compilación y la ejecución con un fichero .bat como este (y de forma similar en Linux):

```
@echo off
inform %1.inf
if %errorlevel==0 goto ejecuta
goto fin
:ejecuta
pause
frotz %1.z5
goto final
:fin
echo Se han encontrado errores. No se ha generado código.
:final
```

Si se guarda como comp.bat, se automatizan los pasos anteriores con:

```
comp hola
```

Puede ser muy ventajoso el utilizar editores integrados que en su parte de *opciones* permiten configurar qué compilador usar, y qué programa quieres utilizar para ejecutar, lo cuál seguramente es mucho más conveniente que trabajar en línea de comando. En la misma página de Zak, existe información y enlaces en este sentido. Pero trabajando en línea de comando, hay que tener en cuenta que no es posible utilizar un editor de MS-DOS, ya que MS-DOS y Windows (inform es una aplicación Win32, aunque se utilice desde línea de comandos) tienen juegos de caracteres diferentes.

3. Comenzando

Siguiendo el ejemplo anterior, `hola.inf`, es posible ver ya algunas características del lenguaje de Inform, el que utilizaremos junto con la librería de aventuras en español.

La primera línea empieza con un '!', en el que se indica un comentario.

A continuación viene la función main, que en ausencia de la librería InformatE!, es como en C, la función a la que se dirige Inform para comenzar la ejecución. La sintaxis no es parecida a la de C, en realidad, además de los corchetes para indicar el comienzo y el final, el nombre de la función se pone a continuación del primer corchete, seguido de un punto y coma. Si la función (que no es el caso) aceptase un argumento, entonces éste se pondría a continuación del nombre de la función, antes del ; y separado por un espacio.

Por lo tanto, ya sabemos escribir una función, para cuando sea necesario.

La orden print no necesita explicación, sirve para imprimir un mensaje en pantalla. Lo que si necesita explicación es el carácter ^ que aparece al final del mensaje: se trata de un cambio de línea forzado (equivalente al '`\n`' de C).

Ahora veremos un ejemplo, pero ya de una verdadera aventura. Hay que tener en cuenta que ahora ya utilizamos InformATE!, y para ella hay que definir unas constantes y rutinas adicionales. Por ejemplo, `main`() permanece escondida gracias a la librería, y en su lugar utilizamos `Inicializar`():

```
!
!  Créditos.
!

! Constantes
Constant Historia "El Abismo";
Constant Titular "^Cruzando el Abismo.^";

! Número de serie (si no se define se aporta uno por defecto)
Serial "092000";

! Constantes para modificar el comportamiento del parser
Constant PUNTUACION_MAX 1;
Constant ADMITIR_COMANDO_SALIDAS;
```

```
! Incluir la librería (en parte)
! Estas líneas están presentes en cualquier aventura
Include "EParser";
Include "Acciones";


! Definición de localidades

!----------------------------------------------------------------------
!      Localidad Abismo
!----------------------------------------------------------------------

Object  elAbismo "Al borde del abismo."
    with irrelevante 'rocas' 'roca',
    with descripcion
       "A tus pies, se abre el abismo insondable. Te encuentras
        de pie en un saliente de roca, de espaldas a la pared sur
        del mismo. Al otro lado, el lado norte, un saliente de roca
        gemelo al que pisas ahora, da paso a la caverna que se abre
        al exterior. En medio de los salientes, la nada.",
    al_n "¡No puedo cruzar el abismo!",
    al_e "Las escarpadas rocas te cortan el paso.",
    al_o "Las escarpadas rocas te cortan el paso.",
    al_s "¡Pero si la pared de roca está a tu espalda!",
    has    luz;


! Después de las localidades, incluimos la gramática
! Este include siempre está aquí en todos los programas

Include "Gramatica";


!----------------------------------------------------------------------
!      Inicialización
!----------------------------------------------------------------------

[ Inicializar;
       localizacion = Abismo;
       "^^^Te adentraste en la cueva del tesoro, y ... recuerdas
        perfectamente como al llegar a la cámara secreta del rey,
        activaste una trampa que podría haber sido mortal ...^
        Buscabas la riqueza, pero ahora te conformas con salir vivo ... ^^";
];
```

4. Perfilando la aventura

En primer lugar, verás que al igual que en C, se incorporan librerías utilizando la orden '*Include'* (sin el '#'). También apreciamos que los comentarios se marcan con el símbolo '!', es decir, la línea que empieza con '!' no es interpretada por el compilador. Los tres *include's* presentes son los necesarios para la librería InformATE!. También vemos la orden '*Constant'*, que declara unas constantes. En concreto, definimos el título de la historia, un pequeño subtitulo, y dos constantes fijas que sirven la primera, para poner el tope de puntuación alcanzable por el jugador, y la segunda para indicar al compilador que vamos a utilizar el comando salidas. Basta eliminar o poner un '!' al comienzo de esas líneas para no permitir estos extras. Con "serial", definimos el número de serie de la versión de la aventura. Además, con PUNTUACIÓN_MAX señalamos a la librería el tope de puntos que se pueden conseguir en la aventura. Entonces, incluimos "acciones" y "eParser". El segundo es obviamente el parser español, mientras el primero carga las reacciones por defecto para muchas acciones. Por ejemplo, si el jugador teclea "salta" en esta cutre-aventura, ésta responde "saltas en el sitio y no consigues nada de nada" (bueno, más o menos), a pesar de que no hemos previsto ninguna acción (todavía) llamada *salta*.

A continuación, viene un elemento totalmente desconocido: un Object, un objeto. Este objeto es una habitación, y no hay una distinción realmente clara entre objetos habitación y cualquier otro objeto. Las diferencias son más bien discretas: por ejemplo, las habitaciones suelen tener las propiedades al_n, al_s... (al norte, al sur ...). Además, son asignadas a una variable global llamada '*localizacion'* que lleva la localidad actual. Por otra parte, el resto de los objetos de la aventura suelen estar dentro (pertenecen a) una habitación, mientras una habitación no tiene un objeto que la contenga. Este concepto de un objeto conteniendo al otro lo veremos un poco más adelante.

Para los lectores que conozcan la notación de la programación orientada a objetos, en Inform tenemos atributos y propiedades. Los primeros se declaran en la línea '`has`', y los segundos bajo el epígrafe *with* o *private*, según sean éstos públicos o privados. Tomando la nomenclatura de objetos en

C++, con '**has**' definimos ciertos atributos para el objeto que proporciona la librería, mientras que con '**with**' definimos datos miembro (o atributos) y funciones miembro (métodos), que en inform son propiedades-.

Empecemos por los atributos: al final vemos `has luz`. Esto indica que el objeto tiene luz propia Si no la ponemos, entonces el jugador no puede ver nada. A no ser que él lleve un objeto que de luz[1], claro.

Al principio, vemos un elemento importante: **with**. Con **with** se definen las propiedades del objeto, mientras que con **has** se definen sus atributos. La parte **with** es la más importante, pues es donde definiremos a dónde se va al ir al norte, que descripción aparece cuando se entra en la localidad, ... etc.

'*descripcion*' es una propiedad utilizada para guardar la cadena que se mostrará por pantalla cuando el jugador examine el objeto, o en este caso, al ser una habitación, cuando el jugador entre en ella. Por otra parte, *al_n, al_s, al_e, al_o* son propiedades que indican que debe hacerse cuando el usuario desea moverse a esas direcciones. En este caso, se ha dispuesto que se visualicen cadenas, pero si en lugar de las mismas se indica un nombre de otro objeto habitación, el jugador se verá trasladado allí al teclear la dirección. Si es una función, la función es ejecutada.

Irrelevante es una propiedad que guarda una lista de palabras. Estas palabras se delimitan con comillas simples, y son en concreto elementos que al ser referenciados por el jugador, provocan un mensaje similar a "`Eso no tiene importancia`". En concreto, si el jugador intenta ir al este o al oeste, obtendrá el mensaje "Escarpadas rocas cortan tu paso." Lo primero que se le ocurrirá al jugador es 'examina rocas', pero al obtener el mensaje "`Eso no tiene importancia`", le quedará claro que "por ahí no van los tiros".

Finalmente, está la función 'inicializar' que con InformatE! cargada, es la función principal. La utilización más común es la de ofrecer una pequeña introducción, y poner la variable localización con el valor de la primera localidad. En este caso, la única: el abismo.

Bien, como en "Cacahuete, Sal, y Aceite" [9], es interesante colocar un cartel que le de alguna pista al jugador. No es conveniente que el cartel se pueda coger, por lo que el objeto deberá llevar la propiedad "estático", y sólo es necesario para que al examinar obtengamos el mensaje "`Ten Fe: pero que no sea ciega`".

```
Object -> cartel "cartel podrido"
with
        nombre 'cartel' 'aviso',
        adjetivo 'podrido' 'mohoso',
        descripcion
                "El cartel indica: 'Ten Fe: pero que no sea ciega.'^",
        inicial
                "Aquí hay un cartel medio podrido por la
                 humedad.",
has estatico;
```

Este código lo añadimos justo después del objeto abismo. La flecha indica que pertenece al objeto abismo, que está dentro de él.

Las propiedades nombre y adjetivo, guardan una lista de nombres y adjetivos que es posible utilizar para referirnos al cartel: cartel podrido, aviso podrido, podrido aviso ... etc.

La descripción, ya la conocemos. Es la cadena a mostrar cuando se hace un '`examinar`' del objeto.

En cuanto a inicial, es el texto que se muestra cuando el jugador describe la localidad. Si se indica aparece un soso "puedes ver un cartel podrido.".

Como hemos puesto la propiedad 'estático', veremos el mensaje "está fijado al sitio" si el jugador intenta cogerlo, lo cuál viene muy bien teniendo en cuenta nuestras intenciones.

Es posible añadir más objetos, para enriquecer la aventura. Es apropiado añadir un abismo insondable.

```
Object -> negroabismo "abismo insondable"
with nombre 'abismo',
     adjetivo 'negro' 'insondable',
     descripcion
           "El abismo es negro e insondable. Muy profundo. Da miedo.",
has oculto estatico;
```

Con la flecha, entre object y negroabismo, se indica que este objeto está en la localidad abismo. En realidad, sólo se indica que está dentro de otro objeto, pero en este caso, el objeto "padre" es una localidad. Pero podría ser otro tipo de objeto. Se ha clasificado como 'oculto', porque no es necesario que aparezca "`aquí hay un abismo insondable`" cada vez que el jugador teclee '`mirar`', y tampoco

[1] Pero en esta *cutre-aventura* no lo hay, desafortunadamente. Se supone que en esta cueva hay iluminación de algún tipo.

sería correcto que apareciera “`has cogido un abismo insondable`” si al jugador se le ocurre teclear “`coger abismo`”.

Además, estás en un saliente de roca, hecho que nos interesa remarcar al jugador (veremos un poco más adelante que en el saliente habrá arena, que será importante para el jugador):

```
Object -> saliente "saliente de roca"
with
     nombre 'saliente',
     adjetivo 'rocoso',
     descripcion
           "El suelo del saliente está lleno de montoncitos de arena.^",
     inicial
           "Estás de pie sobre el saliente de roca.",
has estatico;
```

Y cómo el jugador es muy intrépido y estaba buscando un tesoro, aparecerá en su bolsillo un mapa del tesoro que no sirve absolutamente para nada, en cuanto al desarrollo del juego.

```
Object mapa "mapa del tesoro"
with
    nombre 'mapa' 'tesoro',
    descripcion
          "El mapa que te ha llevado hasta aquí.^
           Desearías no haberlo encontrado nunca."
;
```

Esta vez, no se incluye la flecha porque el objeto no está dentro de nada: bueno, en realidad querremos que esté dentro del objeto jugador, pero la forma de hacer esto es poner, en la rutina de `Inicializar`, una línea como (antes del último mensaje):

```
move mapa to jugador;
```

Esto es aplicable a cualquier objeto, y permite cambiar la posición de un objeto en la jerarquía en cualquier momento. La flecha, sirve para indicar pertenencias iniciales, aunque también se pueden evitar las flechas y hacer un montón de move’s en la rutina de inicialización –al gusto de cada uno-.

Ahora es necesario que pensar en terminar esta aventura: ahora mismo tenemos un abismo y un cartel, pero esto no es suficiente. Vamos a permitir resolverla igual que en “Cacahuete, Sal y Aceite”: echando arena en el abismo que nos permitirá darnos cuenta de que hay un puente invisible.

Así que, en la localidad elAbismo, es necesario incluir una variable que permitirá saber y marcar cuándo el jugador ha lanzado la arena al abismo. Esta variable no es más que un atributo, un dato miembro del objeto:

```
object elAbismo
         with
        arena 0,
               ! Aquí va el resto del objeto (lo que había antes)
;
```

Es decir, tenemos un campo llamado arena con valor inicial 0. Quizá el lector piense que debería haber un signo ‘=’ entre arena y 0: el autor de este artículo está de acuerdo.

Por tanto, es obligatorio completar la aventura, con algún detalle más. Por ejemplo, si el jugador se mueve al norte de la habitación –elAbismo- en la que nos encontramos, se visualiza un mensaje que indica que no se puede cruzar el abismo. Sin embargo, esto no es lo más apropiado. Al ir al norte desde la pantalla en la que está el jugador, debe interpretarse que queremos cruzar el abismo. Para ello, tenemos que añadir un nuevo verbo a nuestra gramática: cruzar “algo”. En inform, tenemos cruzar como verbo admitido por la gramática: pero, además, necesitamos indicar que queremos cruzar una cosa. Esto son las ‘*acciones*’.

Desde un punto de vista de programación orientada a objetos, las acciones son mensajes a objetos. Pero son un tipo especial de mensajes, son manejados por los métodos antes y después. Así, si el jugador teclea ‘cruza el abismo’, el objeto abismo recibirá la acción ‘cruzar’. La propiedad antes y después pueden manejar este mensaje.

Esto, se indica después de incluir la gramática en el fichero de la aventura: es necesario incluir unas líneas, como las siguientes:

```
Extend 'cruza' first
      * noun -> cruza;
```

Estas líneas deben situarse justo después del *include* de la gramática. Con esto, cuando encuentre el verbo cruza o cruzar y un nombre que haya sido definido en el juego, se disparará la acción ‘cruza’. La acción no tiene porqué llamarse como el verbo, es una cuestión de mayor claridad. Toda acción debe tener asociada un comportamiento por defecto, que en Inform toma la forma de una función con el mismo nombre de la acción más el sufijo ‘Sub’. Por ejemplo, esta puede ser la de cruza:

```
[cruzaSub;
       "No he entendido demasiado bien qué quieres cruzar ...";
];
```

La rutina CruzarSub solo se ejecuta si el parsing ha tenido éxito, y por tanto se ha intentado cruzar sobre un objeto definido en el juego, y la acción cruzar se ha generado, y se ha notificado al objeto con un mensaje a través de su propiedad-rutina "antes", pero el objeto ha ignorado este mensaje (por ejemplo porque no se ha especificado propiedad-rutina antes). Esto ocurriría por ejemplo si el jugador pone `cruza cartel` o `cruza arena`. En ambos casos, los objetos son válidos y por tanto se genera la acción cruzar, pero ni el cartel ni la arena tienen prevista una respuesta para Cruzar en su rutina Antes, y por tanto se ejecutará la rutina por defecto CruzarSub.

Ésta es la respuesta que el jugador recibirá cuando intente cruzar cualquier objeto del juego, salvo el abismo que se ocupa por sí mismo de esta acción. Por si te sirve de algo, cuando CruzarSub llega a ejecutarse, la variable global `uno` contiene el objeto que va a ser cruzado, por lo que puedes usarlo como parte del mensaje: `"No tiene sentido tratar de cruzar ", (el) uno, ".";`

La idea es darle a entender al jugador que no tiene sentido en general, o al menos para el juego, el intentar cruzar algo que no sea el abismo.

Pero también se debe tener en cuenta que el jugador quiera saltar el abismo, y no cruzarlo. Podemos suponer que en el contexto de nuestra aventura, saltar es lo mismo que cruzar el abismo, así que en donde se puso la línea de gramática de 'cruza', se puede añadir lo siguiente:

```
Extend 'salta' first
        * noun -> cruza;
```

Fíjese el lector que en este caso, no estamos creando un verbo nuevo, sino que simplemente estamos extendiendo uno que ya existe. Además, nuestras líneas de gramática se pondrán antes de las que ya existían, utilizando 'first'. Esto es una forma de asegurar que lo que nosotros deseamos permitir en el juego va a poder hacerse antes de las acciones por defecto. Aprovechamos para generar la acción `cruzar` en el caso de que se intente `saltar`, que es lo que interesa.

Ahora podemos modificar la dirección norte la localidad abismo, porque está claro que si vamos al norte, queremos cruzar el abismo:

```
al_n [; if (self.arena~=1)
                "¡No puedo cruzar el abismo!";
        else <<cruza negroabismo>>;
],
```

Efectivamente, `~=` es el equivalente a `!=` de C, y por tanto el contrario de `==` en ambos lenguajes.

Es perfectamente legal que al_n sea otra localidad, un mensaje, o como en este caso, una función. La función comprueba si ya hemos tirado la arena y por lo tanto, visto el puente.

Un punto importante es poner *self* en el *if*. Self significa, en inglés 'a sí mismo', así que simboliza al mismo objeto que está ejecutando el código. Es necesario indicar `self.arena` para referirse a arena, a pesar de que arena es una propiedad que está definida en el mismo objeto que este código. Esto puede resultar chocante para el programador de C++, más aún cuando el no indicar *self* antes que arena no siempre provoca un mensaje de error, debido a detalles de *inform* que no tienen cabida en este artículo.

Si la propiedad arena estuviese en otro objeto, entonces se antepondría el nombre de ese objeto, un punto y el nombre de la propiedad, al igual que en C++: la diferencia estriba en que cuando el código pertenece al mismo objeto que donde está definida la propiedad, es necesario anteponer self (equivalente a **this** en C++), cuando en C++ no es necesario.

Y finalmente, debemos pensar en cuándo poner arena[2] a 1. Primeramente, es necesario un objeto arena que se pueda coger. Y que cuando se tire, active arena a 1.

```
Object -> arenaloca "arena"
with
     nombre 'arena' 'granos' 'montoncitos',
     adjetivo 'gordos' 'gorda' 'arenoso' 'arenosa',
     articulo "bastante",
     descripcion
           "Montoncitos de arena gorda ...^",
     antes [;
               echaA: if (otro==negroabismo){
                          elAbismo.arena = 1;
                          remove arenaloca;
                          "Al caer la arena al abismo, parte queda sobre una
                           superficie que no habías visto antes debido a un
                           efecto óptico ... ¡es un puente!.^Ahora podrás
                           cruzar seguro.";
                      }
                      else <<dejar arenaloca>>;
           ],
has oculto femenino;
```

[2] Más que nada, porque es la única manera de acabar la aventura.

Así, si tiramos la arena al abismo, ésta desaparece, y muy importante, ponemos la propiedad arena en el objeto elAbismo, a 1. Ponemos el objeto arena oculta, porque se quiere que el jugador tenga que examinar el saliente para encontrarla, no que se le informe de su presencia al hacer la descipción de la localidad.

La acción echaA se recibe en la rutina miembro `antes`(), y se ejecuta esta rutina antes de que se ejecute la acción por defecto para echaA. Además, tenemos las variables globales `uno` y `otro`, que simbolizan los objetos sobre los que se ejecuta la acción. La primera de ellas contiene una referencia a *arenaloca*, claro (ya que es la que recibe la acción), mientras la segunda contiene el receptor de lanzar la arena, que es el abismo.

A continuación, el código fuente completo de esta mini-aventura. Es muy parecida a la segunda prueba de Cacahuetes, Sal, y Aceite [9], y es posible bajársela en la misma página que ésta última, como tutorial de Inform.

```
!
!  Créditos.
!

! Constantes
Constant Historia "El Abismo";
Constant Titular "^Cruzando el Abismo.^Febrero de 2001^";

! Número de serie
Serial "022001";

! Constantes para modificar el comportamiento del parser
Constant PUNTUACION_MAX 1;
Constant ADMITIR_COMANDO_SALIDAS;

! Incluir la librería (en parte)
! Estas líneas están presentes en cualquier aventura
Include "EParser";
Include "Acciones";

! Definición de localidades

!-------------------------------------------------------------------------
!      Localidad Abismo
!-------------------------------------------------------------------------

Object  elAbismo "Al borde del abismo."
    with irrelevante 'rocas' 'roca',
         arena 0,
    with descripcion
       "A tus pies, se abre el abismo insondable. Te encuentras
       de pie en un saliente de roca, de espaldas a la pared sur
       del mismo. Al otro lado, el lado norte, un saliente de roca
       gemelo al que pisas ahora, da paso a la caverna que se abre
       al exterior. En medio de los salientes, la nada.",
    al_n [; if (self.arena==1)
               "¡No puedo cruzar el abismo!";
             else <<cruza negroabismo>>;
         ],
    al_e "Las escarpadas rocas te cortan el paso.",
    al_o "Las escarpadas rocas te cortan el paso.",
    al_s "¡Pero si la pared de roca está a tu espalda!",
has    luz;

Object -> saliente "saliente de roca"
with
     nombre 'saliente',
     adjetivo 'rocoso',
     descripcion
           "El suelo del saliente está lleno de montoncitos de arena.^",
     inicial
           "Estás de pie sobre el saliente de roca.",
has estatico;

Object -> negroabismo "abismo insondable"
with nombre 'abismo',
     adjetivo 'negro' 'insondable',
     descripcion
           "El abismo es negro e insondable. Muy profundo. Da miedo.",
    antes [;
          cruza: if (elAbismo.arena==0)
```

```
                {
                    banderafin = 1;
                    "Intentas cruzar el abismo de un salto pero ... fracasas.";
                }
                else {
                    banderafin = 2;
                    "Cruzas el abismo por el puente que vislumbras vagamente ...";
                }
    ],
has oculto estatico;

Object -> cartel "cartel podrido"
with
       nombre 'cartel' 'aviso',
     adjetivo 'mohoso' 'roñoso' 'viejo' 'podrido',
       descripcion
              "El cartel indica: 'Ten Fe: pero que no sea ciega.'^",
       inicial
              "Aquí hay un cartel medio podrido por la humedad.",
has estatico;

Object -> arenaloca "arena"
with
     nombre 'arena' 'granos' 'montoncitos',
     adjetivo 'gordos' 'gorda' 'arenoso' 'arenosa',
     articulo "bastante",
     descripcion
           "Montoncitos de arena gorda ...^",
     antes [;
               echaA: if (otro==negroabismo){
                          elAbismo.arena = 1;
                          ++puntuacion;
                          remove arenaloca;
                          "Al caer la arena al abismo, parte queda sobre una
                           superficie que no habías visto antes debido a un
                           efecto óptico ... ¡es un puente!.^Ahora podrás
                           cruzar seguro.";
                      }
                      else <<dejar arenaloca>>;
           ],
has oculto femenino;

! Después de las localidades, incluimos la gramática
! Este include siempre está aquí en todos los programas

[echaASub;
   "No he entendido demasiado bien qué quieres echar y en dónde ...";
];

[cruzaSub;
   "No he entendido demasiado bien qué quieres cruzar ...";
];

Include "Gramatica";

!Definiciones de la gramática para este juego

Extend 'cruza' first
      * noun -> cruza;

Extend 'salta' first
      * noun -> cruza;

Extend 'tira' first
      * held 'a' noun -> echaA
      * held 'al' noun -> echaA
      * held 'en' noun -> echaA;

Extend 'lanza' first
      * held 'a' noun -> echaA
      * held 'al' noun -> echaA
      * held 'en' noun -> echaA;


Extend 'echa' first
      * held 'a' noun -> echaA
      * held 'al' noun -> echaA
```

```
      * held 'en' noun -> echaA;

!-----------------------------------------------------------------------
!       Inicialización
!-----------------------------------------------------------------------

Object mapa "mapa del tesoro"
with
    nombre 'mapa' 'tesoro',
    descripcion
          "El mapa que te ha llevado hasta aquí.^
           Desearías no haberlo encontrado nunca.",
    antes [;
          echaA: if (otro==negroabismo)
                 {
                    remove mapa;
                    "El mapa cae al abismo y se pierde en sus profundidades.";
                 }
    ]
;

[ Inicializar;
        localizacion = elAbismo;
       move mapa to jugador;
       "^Te adentraste en la cueva del tesoro, y ... recuerdas
        perfectamente como al llegar a la cámara secreta del rey,
        activaste una trampa que podría haber sido mortal ...^
        Buscabas la riqueza, pero ahora te conformas con salir vivo ... ^^";
];
```

5. Notas

[1] El compilador Inform para la máquina Z, de Graham Nelson.
http://www.gnelson.demon.co.uk/inform.html

[2] El traductor de las librerías de Inform al español, InformatE!. spinf_2000@yahoo.com
Lista de distribución sobre InformatE! http://www.egroups.com/lists/informate
Página de InformatE! http://www.geocities.com/TimesSquare/Fortress/9939/index.html

[3] DocumentatE! Es la traducción (adaptada a las características de InformatE! en español) del manual para Inform de Graham Nelson. Se puede encontrar en las mismas páginas que InformatE!.

[4] El frotz para diferentes plataformas también en las páginas de InformatE!.

[5] Siempre es posible convertir una máquina en otra. Esto lo demostró el matemático Alan Turing, utilizando su Máquina de Turing. La máquina de Turing es un concepto matemático del que el autómata de estados finito es un caso particular. El resultado en concreto es, dado que, desde un punto de vista matemático, la máquina de Turing simula perfectamente una computadora, y dado que se demuestra que siempre puede construirse una máquina que acepte la codificación de otra máquina y una entrada para esa máquina, y genere la salida adecuada, cualquier computadora puede simular a otra.

[6] Iznoguz puede bajarse en http://www.geocities.com/TimesSquare/Fortress/9939/index.html

[7] Por cierto, encontrarás enlaces a las páginas anteriores en la página del CAAD. http://pagina.de/caad

[8] Para descomprimir los paquetes, utilícese el descompresor Zip. En el caso de Windows, es posible utilizar pkunzip o WinZip, mientras que en el caso de Linux, unzip.

[9] Fantástica aventura ;-) que se puede bajar de la parte de concursos del web del CAAD. (http://pagina.de/caad) ó también de http://usuarios.tripod.es/elarquero, e incluso de las páginas de Zak.