# The “Which Authoring System is Better” FAQ

Bob Newell (bnewell@gobblernet.sputnix.com)

Revision 0.1
February 17, 1996

## Contents

# 1 The Purpose and Nature of This FAQ

For the prospective author of Interactive Fiction (IF) there is now a wide choice of authoring languages. While it is immediately obvious that not all these languages are created equal, nevertheless the process of choosing a language can be a daunting task for the new IF author, and is not always simple even for the experienced author.

The purpose of this FAQ is to describe at a high level most of the major and some of the minor authoring languages, presenting a framework for evaluation, and some subjective comparison. This FAQ will not choose a system for you, but instead has the archetypical aim of having you-go away with sufficient information to make a decision which is a tad more informed.

In nearly all cases, the creators of authoring systems are well respected and highly deserving of this respect. Comments and criticisms directed at authoring systems within this FAQ are intended to be constructive and no disrespect for, or lack of appreciation of, the creator of the authoring system is in any manner intended.

While numerous people have helped immensely in the production of this document, I take full and complete responsibility for any deficiencies in its content. No opinion expressed herein should be assumed to be or not be the opinion of any individual contributor to the FAQ, and the presence of their name in the credits is not an indication of their agreement or disagreement with any statement made in the FAQ.

This document is a total rewrite of my older *Which is Better, TADS or Inform?* FAQs, and for the most part replaces and supersedes the older documents. That document went into excruciating detail about things such as operator sets, library function calls, and other such matters which, while they are important, will not greatly influence the more high-level process of choosing a system. Prospective authors who really need to compare the small details in an in-depth manner should refer to the documentation for the individual systems and make their own comparisons.

# 2 What Authoring Systems are About

Drop in on the main Usenet conference for IF authors, `rec.arts.int-fiction`, and you'll find that authoring systems are a hot topic. From time to time someone, usually but not always a relative newcomer to the newsgroup, will post a question about writing interactive fiction in Lisp, C++, Basic, or even Pascal or Fortran. Almost always a response comes back, "Haven't you checked out authoring systems X, Y, or Z?"

Indeed, while it is entirely possible and feasible to write IF in a procedural (or even non-procedural) 3rd generation language, object-oriented or non object-oriented, there seems today to be little point in so doing. Specialized languages, created for the main purpose of IF authoring, save the prospective author a great deal of drudgery, primarily in the area of parsing and natural language processing. Except as an experiment or as an exercise in programming, new authors quickly turn to one of these purpose-built languages for the creation of their proposed IF work.

The first purpose of the authoring system, then, is to make the author's life easier, and much of the discussion in this document will revolve around just how, and how well, this takes place. A second purpose of the authoring system, and probably no less important, is to make life easier for the prospective *player* of the finished piece. (The term *player* will be used in preference to the term *reader* since the works of interactive fiction treated here are invariably presented in the context of a game.) A good natural language interface and an effective parser make the work a more pleasant and appealing experience to the player.

Further to this, and of a more technical nature, the end product or output of applying the authoring system to the work at hand is some sort of *game file* or *story file* which is then run in some manner by the prospective player on some computer system. The wider the type and variety of computer systems upon which the game or story file will operate closely defines the possible audience for the work. Therefore, an authoring system which produces game files which only run on Cray supercomputers will be of considerably lesser value than a system which produces story files workable on all manner of PC, Macintosh, Commodore, Acorn, and Sinclair.

So, in summary, an IF authoring system helps the author create the work; helps the player experience the work; and defines the possible audience of the work. These major factors, and a number of minor ones, will form the basis of our evaluation.

# 3 An Evaluation Framework

Based on a very much subjective analysis of the traffic on the newsgroup, `rec.arts.int-fiction`, and some limited feedback on an early draft version, I present a set of some nineteen suggested criteria for evaluating IF programming systems. I've weighted each criterion for importance on a 1 to 5 scale. In an exercise as approximate as this, I see no point in expanding the scale to allow for finer gradations of meaning.

It would be very tempting for the reader, at the end of the paper, to multiply weightings and ratings for individual authoring systems to come up with a composite score and declare one system to be the "winner" based on these scores. This would be fallacious, in that the *relative* importance of the categories to one another have not been evaluated. Two categories of weight 5, while both are critical, can not be assumed to have the same overall impact on the usefulness or desirability of a given authoring system. Furthermore, it cannot be argued that the categories listed form a complete, comprehensive, orthogonal set which would even be an appropriate basis for computing an overall score.

The reader is therefore strongly discouraged from the computation of composite scores, and instead encouraged to look into individual ratings categories and so determine for oneself what is important in any particular situation or environment.

The evaluation category weightings have the following approximate meanings:

**5 Critical.** The usefulness of the system will be severely compromised or destroyed if this category does not measure up.

**4 Very important.** The system does not fully stand or fall based on this factor, but usefulness will be significantly restricted if it does not measure up.

**3 Important.** A system rating poorly in this area will still be useful but its quality and ease of use will be somewhat restricted.

**2 Desirable.** This area is one that should measure up but is less critical to usefulness or usability.

**1 Nice.** An area that is nice-to-have as a convenience for added utility or ease of use, but is by no means a necessity in a fully usable system.

## 3.1 Categories and Weightings

Here are the nineteen categories and weightings which will be used throughout our evaluations:

1. **Documentation, Weight 5.** A poorly documented system is nearly useless, whereas a system has often been chosen over another merely due to the quality of the documentation.

2. **Ease of Use, Weight 5.** This is a catch-all category which takes into account the "feel" of the system to the author and the "comfortableness" of use of the system. "Uncomfortable" systems can be virtually useless. As this category is rather large, I'll be discussing various aspects of it as they arise.

3. **Parser, Weight 5.** As the primary interface to the player of the game, the quality of the parser is critical. A system with a poor parser will be difficult for the author to work with but will annoy the player so quickly that the game will be rapidly set aside. Parser extensibility forms part of the overall parser quality rating as well.

4. **Support, Weight 4.** How well is the system supported by the author of the system? How well maintained is the system? How quickly are bugs fixed or workarounds provided? How much newsgroup support can you expect? For the beginning author, especially, support can often be the crucial difference between a work that can achieve completion and one that is abandoned in frustration.

5. **Language Depth, Weight 4.** How extensive is the language in terms of functions, operators, constructs, etc.? How good and useful are the default libraries? This rating is 4 rather than 5 as depth can be a mixed blessing. A language must have sufficient depth to be useful, and enough additional depth to be easy to use. However, great depth can come at the cost of complexity, ultimately reducing ease of use.

6. **Portability, Weight 4.** Where will the final game run? On how many computer systems, of what size and expense? This is a perennial topic of debate on the newsgroup. It is clearly of major importance since a system producing games running on but few computers will not have much of an audience no matter how good it is.

7. **Run Speed, Weight 4.** If the game runs slowly and ineffectively, the game playing audience, often a fickle and argumentative lot, will desert ship quickly. Witness the discussion of *The Legend Lives* in mid-1995 which unfortunately centered for a while not on a fine, artistic work, but on how much trouble it was to run on smaller computers or in certain computing environments.

8. **Literature Library, Weight 4.** Most programmers learn by example. If a large library of source code is available, the system will be easier to learn and use. Browsing source code to find the solution to a given implementation problem is usually efficient and time saving.

9. **Debugging Features, Weight 4.** What facilities are offered the author *cum* programmer for locating and repairing the bugs that appear inevitably in any program longer than two lines? Lack of a debugging facility can make the author's life miserable while a good set of tools can greatly shorten the production time for the piece.

10. **Future Prospects, Weight 3.** What is the outlook for further development of this language in the future? Is the author of the system still actively involved with it? How often do new releases come out, and how substantial are they? How quickly are bugs fixed? In short, is the language living, classical, or dead?

11. **Object Orientation, Weight 3.** How closely does the system follow an object-oriented paradigm? Object-orientation simplifies programming in a number of important ways. To rate highly, a system should approach a "pure" object-oriented paradigm (which I will not attempt to define).

12. **Game Size, Weight 3.** How large a game will the system produce? This factor might have rated higher except for the fact that most systems can produce a reasonable size game- of the kind that most authors have in fact produced. Very large games, requiring years of effort, appear much less often.

13. **Distribution Policy, Weight 3.** What sort of restrictions does the author of the language impose on distribution of games produced with this system? This category would rate higher if it were not for the fact that most distribution policies only concern outright commercial distribution.

14. **Programming Skill Required, Weight 3.** I weight this as a middle of the road criterion since it is either quite important to you or it is not very important at all, depending on your programming skills and experience. For a neophyte programmer this could be a critical criterion, whereas for an experienced programmer, a more "advanced" language will present no barrier. Keep in mind that, despite some representations to the contrary, all of the systems require some programming skill (more on this later) although there are some marked variations.

15. **Cost, Weight 2.** Except in extreme cases, shareware costs have surprisingly not been very much of an issue. For a few people, the free software philosophy has determined a final choice, but this is not generally the case.

16. **Source Code, Weight 2.** Is source code for the system (as opposed to the libraries only) available? This is of direct importance only to dedicated hackers or port maintainers. However this can be an indicator of the future viability of the system, and the ability of the system to be propagated to varied computer systems. Of interest here also is the availability of active porters, which will determine how quickly new releases get propagated to the various computing platforms.

17. **Compiler Speed, Weight 2.** A fast game compiler is nice, of course, but most authors say that it is not a determining factor. In fact, it could be perversely argued that a compiler that is *too* fast might tempt an author to write poorly designed code!

18. **Compiler Platforms, Weight 2.** The number of different computer systems on which the compiler will run, in contrast to the game itself, receives a lot less attention but nevertheless is of interest to those prospective authors with lower-end computers.

19. **Dynamic Object Creation, Weight 1.** Perhaps this does not need a category of its own, but it has been often discussed as a convenience factor. A system will be a little more desirable if there is a means to create and destroy objects during the course of the game, rather than only at compile time. One correspondent pointed out quite rightly, though, that this feature can be at odds with reliability and portability.

## 3.2 Rating the Systems by Category

Each authoring system to be studied will be assigned a rating for each category in the evaluation criteria. I believe the traditional one to ten rating scale will yield an implied precision of rating which is misleading, so I have again chosen a one to five rating scale with the following meanings.

**5 Superb.** A rating at this level means that the authoring system, in this category, is a sparkling jewel. Although still perhaps capable of improvement (for perfection is seldom achieved) the rating implies clear and obvious excellence.

**4 Very Good.** If not a sparkling jewel, certainly a gold coin. The system stacks up very well in this category and leaves no gaps.

**3 Good Enough.** A well-circulated silver coin. While adequate overall in this category, there may be some non-fatal gaps or shortfalls which reduce the luster even while the coin remains of reasonable value. A definite middle of the road rating.

**2 Needs Work.** You can still spend this coin but merchants will give you odd looks. There are serious gaps or shortfalls in this category which merit a decidedly lower rating.

**1 Horrid.** No one will accept this coin in trade. The deficiencies in this category are fatal. Should such a rating occur in a highly weighted evaluation category, the system will be rendered virtually useless.

# 4 The Tiers of Authoring Systems

## 4.1 The Tiers Defined

Following a model used by some rather expensive and high-powered US-based computer industry analysts, and in informal collaboration with Jools Arnold to keep to a consistent categorization, I divide IF authoring systems into three tiers and one additional non-tier.

The **first tier** is that of *world class contender.* The very few entries in this tier are at a stage of development that clearly puts them in a rather elite category of their own.

The **second tier** is composed of a relatively larger number of systems which may be categorized as a *strong candidate.* Authoring systems in this category might be up-and-coming entries with a great deal of promise; a system which was once very popular but has faded for various reasons; or a system of high quality but a lesser following for a variety of reasons.

The **third tier** is composed of a collection of *also-rans.* These systems are categorized as such because they may be very old and outdated; they may be experimental and undeveloped; or, quite bluntly, they may be just plain inferior or non-functional.

The non-tier is the **unprocessed tier** and is made up of relatively new systems for which sufficient time and experience has not accumulated to assign it to one of the three regular tiers. It is expected that an entry will move from here to the second tier if enough interest in the system arises, and the author continues to develop it. (Direct entry into the first tier would be most unlikely.) The system might drop into the third tier if it is largely ignored or if the author does not develop it very far.

Entries in the first tier will receive the most detailed study; entries in the second and unprocessed tiers will receive complete but less detailed treatment; and entries in the third tier will receive only representative or cursory evaluations.

### 4.2 The Population of the Tiers

The **first tier** members are **Inform** and **TADS** (the Text Adventure Development System).

The **second tier** members are **ALAN, Hugo,** and **AGT** (the Adventure Game Toolkit).

**Unprocessed tier** entries are **Archetype** and **GINAS**. **Rexx Adventure** would also go in this category, but I am unable to review it at this time as I don't have an OS/2 system.

In the **third tier** are **OASYS, LADS, Gamescape, ADL, Aventuro, AdvSys,** and a host of others too numerous to remember. LADS, Gamescape, GINAS, and Aventuro will be reviewed here. OASYS, ADL, and AdvSys will be reviewed in a future edition of this FAQ; I've simply run out of time in this first edition.

Graphics-oriented systems such as GTAC, DC and others are not included in either the tier groupings or the study itself.

## 5 The Systems Examined

What follows is a necessarily lengthy discourse, taken authoring system by authoring system, which includes a summary description of the system (at times borrowed shamelessly but with permission from Jools Arnold's *Rec.Arts.Int-Fiction FAQ*) and a rating with explanation in each rating category. Finally, salient or unique features and limitations of the system are described. In only a few instances are projected future developments discussed; I made an arbitrary policy decision in this FAQ to only deal with what exists today.

The impatient might wish to page to the very end where you will find that a table is given which cross-references ratings by category and authoring system.

### 5.1 TADS— Text Adventure Development System

**Release discussed: 2.2.0.5**
**Author contact information:** Mike Roberts `73737.417@compuserve.com`

**Description.** TADS, the Text Adventure Development System, is the creation of Mike Roberts, and has been in existence and has evolved over a period of several years. TADS is an object-oriented system which has a programming "feel" similar to a Pascal/C hybrid.

TADS is indeed a very powerful system, and some very large works have been published or are in progress with TADS. TADS is shareware; the full package, minus printed documentation and debugger are available from the archive site and the other usual places. If you ultimately choose to work with TADS, you should register. No doubt about it.

TADS is rich in feature and function, with a good basic library and a comples "alternative" library, called *WorldClass*, also available. Little has been left out of TADS: there is the equivalent of associative arrays, list-tearing functions, a complete operator set, and much more than can be described here.

**Ratings.** Ratings by category for this system are:

1. **Documentation, Weight 5. Rating: 4.** The TADS manual supplied to registered users is uniformly considered to be a superb piece of work, with few deficiencies and very high utility. (The TADS documentation supplied with the shareware version is barely adequate to evaluate the product; however, this documentation is not the subject of the present rating.)

   TADS has gone through several updates since the hard-copy registered TADS manual has been produced. The updates have been substantial and far-reaching, making the product a ground-breaking leader on several fronts. Supplementary documentation has been included with each revision in the form of text files accompanying the release. This documentation has been of good quality, but the net result is that the documentation has become disjointed and scattered among the hard-copy manual and the text files.

   I understand the expense and difficulty in reissuing the hard-copy manual; at the heart of the matter lies the fact that the hard-copy manual is the major registration incentive. But this does not make the difficulties go away.

   At the more fundamental levels, the existing manual will more than suffice, and the prospective author will get a good head start by studying this clearly-written and extraordinarily helpful work. It is when you want to access the newer, more sophisticated and original features of TADS that you wish for unified documentation.

   I cannot offer an easy solution. Perhaps at some point Mike Roberts, the creator of TADS, might wish to update the manual and offer it to existing registered users for a reasonable price (as he has done with supplementary documentation in the past). I'd suggest that release 3 of TADS, whenever that might come, might be a good point to do this.

   Additionally, I do not feel the TADS documentation will "work" for a neophyte programmer. Of course, I do not believe TADS is for the neophyte in any case; see my comments in the appropriate ratings section.

One other slight personal issue I have with TADS documentation is that you can't keep it all on-line. I often like to have my source code in one editor window and the manual in another; you simply can't do this with something available only in hard-copy.

2. **Ease of Use, Weight 5. Rating: 4.** I must qualify this rating by stating that, *with sufficient study*, ease of use of TADS is quite good. It is not reasonable to expect a powerful and complex system such as this to be easy to use without doing your homework first.

   There are some peculiarities in TADS that frustrate the newcomer but become second nature with experience. TADS has, and indeeds requires, a plethora of verification methods which are applied when a verb is combined with a direct and/or indirect object. For instance, "attack monster with sword" requires that the nouns "monster" and "sword" have methods defined which specifically allow (or disallow) usage in the combination described with the verb "attack."

   Such peculiarities aside, the clean object-oriented structure of a TADS program enhances ease of use to a very great extent. Even fairly complex objects and actions can be defined quite quickly and readily by someone with a little TADS experience. Objects lend themselves readily to reuse, even from program to program, and indeed a collection of such reusable, and useful, objects can be found at the interactive fiction archive site.

   TADS is enough like and not like Pascal, and like and not like C, to be a bit confusing at the start. Although the most recent version allows for a more strictly C-like syntax, the beginner is best advised to work through the initial confusion. It doesn't take long to get comfortable with this language.

3. **Parser, Weight 5. Rating: 4.** The parser is built-in and very good, understanding a wide variety of constructs and long sentences. Multiple objects, pronoun antecedents, indirect objects, varied word order, compound sentences, and much more, are easily interpreted by the parser. A number of methods and functions are present by which default parser behavior can be modified or overridden, including one routine that allows you to fully substitute for the parser should you so wish (this should be pretty rare).

   The down-side of the parser is that it is built into the source code, not the libraries; and further, with source code not generally available, you're going to have to live with the parser and ancillary functions as-is. Some authors, particularly those wishing to write in languages other than English, have stated that this is a problem to varying degrees. For most authors, and particularly for the prospective new author, this should be a minor concern until a very advanced level is reached— or unless you are not writing in English, in which case the issue is a little more serious (but hardly fatal).

4. **Support, Weight 4. Rating: 5.** Support for TADS can be obtained in three ways: from the author by e-mail; via the newsgroup; or via the TADS support BBS in California. E-mail support has been reported to be a little spotty due to Mike Roberts' busy schedule. The support BBS has been successful for the relatively few that have used it, although there are some problems here- you can't call at rates higher than 2400 bps, and it's a long distance or international call for most people. News group support, however, is really good, as there is enough developed TADS expertise among people willing to help out that a good source of advice is readily available.

   Discussion of TADS in recent months has been significantly less than discussion of Inform. Whatever may be the reasons, this does not imply a decline in the fortunes of TADS. In fact, the 1995 IF contest drew as many TADS entries as Inform entries. TADS expertise, at a high level, remains readily available on the newsgroup.

5. **Language Depth, Weight 4. Rating: 5.** Language depth is extraordinary. About the only thing that seems to be missing is floating-point numbers, and these are likely to be of limited utility in an IF piece (suitable workarounds are usually at hand). TADS functions for string and text processing are very complete, perhaps approaching the completeness of the Perl language. Array and list manipulation is provided. Available function calls cover just about everything you might need, from determining if an object has a particular property to changing the default behavior of the parser. TADS has a large number of functions which deal with direct and indirect objects in a sentence. These are a mixed blessing, as will be discussed later.

   Again, with language depth comes inevitable complexity. Having all these features available means, if you wish to use them, having to learn how to do so. Yet, the features are layered well enough that for the most part, the prospective author can learn them one at a time, as they are needed.

   One thing that sets TADS apart from the other systems is that there is a choice of "standard" libraries. The libraries supplied with TADS are generally considered to be quite good, reasonably adaptive and flexible, and in any case can have components easily substituted or modified. However, an alternative set of libraries exists, called *WorldClass*. These libraries, written by Dave Baggett, are very complex and extensive, providing for improved environment modeling and character interaction. It is an understatement to say that these libraries require some study; and, since they are very large, they do significantly impact both the run-time performance and the run-time portability of the completed game. Nonetheless, a master-work such as *The Legend Lives* would have been somewhat more difficult to produce without these powerful but demanding alternative libraries.

6. **Portability, Weight 4. Rating: 4.** TADS runtimes for a number of Unix systems, the Mac, and the PC are all up-to-date. Support for the Amiga has lagged, and has not been present for other systems.

   In the old days of the pure "TADS vs. Inform" debates, TADS was down-rated by some newsgroup correspondents because it would not run on some of the less common computer systems, which, despite having small market share, still represented a substantial number of potential players. I think this case was overstated, but nevertheless I rate TADS as 4 for portability rather than 5 because other major systems do support more platforms. However, we really must remember that a TADS game can be played by a good 98 or 99 percent of the computing public, should they have an interest.

7. **Run Speed, Weight 4. Rating: 3.** TADS run-time speed is fine for smaller games built with the standard libraries, on most "reasonable" computing platforms. Noticeable problems result, however, with larger games, especially games built with the "World Class" libraries (although this latter comment is based only on the single example available). Large libraries provide large object sets and vocabularies, and while these add immeasurably to the depth of the system it is not without a price. Some players using lower-end machines have called large games "unplayable" or "unbearably slow." I think these criticisms are too extreme and perhaps reflect a tendency toward impatient play rather than measured enjoyment of an IF work. Nevertheless, there is some truth here. The TADS run-time on larger works is very clearly the slowest in its class.

8. **Literature Library, Weight 4. Rating: 5.** Extensive literature exists in TADS (there is a separate document detailing this which appears on the newsgroup from time to time). Much literature is available in source code format, ranging from *Colossal Cave* to *Legend* itself. There is enough for the prospective author to be able to find an example for just about any sort of programming construct or problem. TADS' strong literature library is a very important distinguishing factor.

9. **Debugging Features, Weight 4. Rating: 4.** The TADS source level debugger, supplied with the registered version to PC and MAC users, is everything you would expect of a debugger. Breakpoints, conditional breakpoints, expression evaluation, variable modification, branching, etc., are all supported smoothly.

The rating would be even higher were it not for two problems which are more logistical than technical. The debugger is not available for other platforms (less serious point), but apparently has not been kept up to date for the major platforms (serious indeed).

For registered users, picking up a new debugger version (which must match the compiler version) has been a matter of a long-distance call to the TADS support BBS, Hi-Energy BBS, in California. In recent times however, more current versions of the system have been placed on the Internet and the BBS lagged. I still do not have a debugger for TADS 2.2.0.5, which is the latest release.

Perhaps a better method of updating and distribution can be worked out. This is the only real flaw in the debugger, and it has nothing to do with its utility or technical merits.

10. **Future Prospects, Weight 3. Rating: 3.** I am sure to get criticism about this relatively low rating for TADS future prospects. While TADS is very much alive, very viable, widely used, and well supported, new releases have become more and more infrequent. This, I am sure, has to do with Mike Robert's busy schedule more than with any lack of interest on his part. Yet, this, coupled with the general unavailability of source code, lead me to the conclusion that future developments could be limited. Mike has projected some features for an eventual 2.3 release, but there is no release date and no one really expects to see it soon.

11. **Object Orientation, Weight 3. Rating: 5.** Object orientation in TADS is as "pure" as one might like (although you can quibble about things like classes and instantiation of a class object being equivalent). TADS supports class definitions, multiple inheritance, overriding, encapsulation of data, messaging in a very general sense, and all the regalia of object orientation.

    Reduced to practical terms, this makes TADS very well suited to the purpose at hand, very much easier to work with, and very much more logical in structure. Mike Roberts deserves the highest praise for the quality of this implementation.

    TADS was, a few years ago, my first experience with object-oriented programming. Predictably, it was a little frustrating until I got the hang of it— more than once I wished for my old, procedural systems— but in the end, I found myself wondering what the problem really was, and why on earth would anyone *ever* want to do it the old way any longer?

    For the prospective new author, there will likely be some learning necessary here, but it will stand one in good stead for a very long time.

12. **Game Size, Weight 3. Rating: 4.** TADS can produce very large works, yet I rate this category just below top level because the large works

carry along significant portability problems which are just now being partially solved.

*The Legend Lives* when it was released was the largest TADS work ever. It used the special *WorldClass* libraries and, on the PC platform, depended on a protected-mode version of the runtime in order to be usable. This runtime was plagued with so many compatibility problems that newsgroup discussion of Legend centered (most unfortunately) on problems with running the game rather than with the quality of the game itself.

It can be argued rightly that DOS compatibility problems are the fault of DOS, Windows, the myriad incompatible memory management systems used to get around the DOS memory usage limitations, and all those problems with which PC users are all too familiar. Nonetheless, the PC represents probably 90 percent of the playing platforms, and the IF runtime system simply *must* deal with this.

A more stable version of the runtime has been released but will not run on sub-386 PC models.

So, TADS can produce very large games, but there will be issues to consider.

Of course, as a practical matter, it is still possible to write a large game (as opposed to an enormous one) within the limitations of the "standard" TADS runtime environment, and this will meet the needs of most authors very readily.

13. **Distribution Policy, Weight 3. Rating: 4.** The TADS license allows unrestricted shareware distribution of any TADS game, which will cover the needs of the majority of authors. Commercial distribution requires a separate arrangement with TADS creator Mike Roberts, which I am told is generally quite easy to arrange.

14. **Programming Skill Required, Weight 3. Rating: 3.** Assigning this rating is an attempt to reflect the reality of TADS measured against its purpose and intent. TADS does not claim to be intended for novice programmers, although I think the documentation understates the amount of knowledge necessary. (Professionals in this field, including myself, tend to assume that non-professionals are a lot more advanced than is really the case.) But TADS does not claim to be a simple or unsophisticated system, and indeed it is not. Quite the opposite is true. TADS is a state-of-the-art, Tier 1 system, and this imposes some definite requirements on the prospective user.

    I do not believe that a non-programmer can be very successful very quickly with TADS. It is not out of the question for TADS to be someone's "first" language; but in that event, the documentation won't do. The documentation implies a certain level of programming knowledge which is quite a bit beyond beginner level.

I don't rate TADS "4" on programming knowledge required because of its complexity for a beginner. Neither do I rate it "2" because a sophisticated language of this type cannot be expected to be elementary and easy.

The prospective TADS user had better bring to the party an intermediate knowledge of C or Pascal (not Basic, thank you). Guru-level knowledge isn't required, but familiarity with programming concepts and more than a little experience using them is highly recommended.

15. **Cost, Weight 2. Rating: 4.** I must admit that in assigning this rating I'm making an implicit judgment of price vs. value. TADS is not free, it is shareware, and the registration is currently US $40.00 plus some shipping charges. For this you receive the excellent but somewhat out-of-date printed TADS manual, and some extra utilities including a source-level debugger, if you are a PC or MAC user. Nearly everyone agrees that the $40.00 asking price for TADS is more than fair, and that Mike Roberts treats customers well and fairly. There are a few people to whom cost is a major factor, whether on philosophical or "impoverished student" grounds, but for most the cost of TADS is a non-issue.

16. **Source Code, Weight 2. Rating: 2.** Source code for TADS is available only upon acceptance by Mike Roberts of your signed non-disclosure agreement, and Mike understandably limits distribution of the source. Source code, for the majority of authors, can be said to not be available.

17. **Compiler Speed, Weight 2. Rating: 5.** I did a series of benchmarks a while back, comparing TADS and Inform for compilation speed on a number of hardware and operating system configurations. TADS compilation was extremely fast even on slower systems, and nearly instantaneous on new, fast systems. (The test case was the ubiquitous port of Colossal Cave.) The compile speed on my Pentium 133 running Linux is literally breath-taking. DOS compile speed is about twice as slow but still blazingly quick.

    (When I first published these results, I received a number of "so what" responses— the point being that run-time speed was far more important than compile speed. True enough; yet, I find a lot of advantage in being able to compile quickly and quickly eliminate syntax errors and the like. Of course, there can then be the afore-mentioned temptation to write sloppily...)

18. **Compiler Platforms, Weight 2. Rating: 4.** I rate this 4 instead of 3 because most of the "world" is covered with DOS and MAC compilers

readily available (and Linux and Sun as well). I recognize the frustration of Amiga users, whose versions have lagged; and I know there are other systems which are not covered. But the mainstream is well supported at this point.

19. **Dynamic Object Creation, Weight 1. Rating: 5.** Recent developments within TADS have made dynamic object creation and destruction, name-changing, etc., very easy to accomplish. Distinguishing between identical objects (lottery tickets, mailboxes, etc.) is also done quite readily.

**Summary Comment.** TADS is mature, quite bug free, stable, and has been in use for some time. Support is excellent, documentation is good, and the language is powerful and highly object-oriented. TADS demands study and some patience in order to unleash its inherent potential. This patience will be well rewarded. The prospective serious IF author will not go wrong with TADS. Not at all.

## 5.2 Inform

**Release discussed: 5.5**
**Author contact information:** Graham Nelson, `nelson@vax.ox.ac.uk`

**Description.** Inform is an interesting creation, in every imaginable way, and arose from a unique concept: Graham Nelson's inspired idea of creating a compiler that would produce Infocom format story files, in a sense calling upon and reviving the greatness of the Infocom era. Along the way, Graham developed Inform to the point at which it is clearly in the premier tier of interactive fiction authoring systems.

Inform's heritage delineates Inform's peculiarities and strengths at the same time. The Infocom story file format imposes limitations; it also ensures high portability and a well-understood run-time environment.

Inform has recently been used to produce works such as *Jigsaw*, which lead IF in new directions and set new standards. Inform, in a rather short time, has evolved from curiosity to a preeminence shared only with TADS.

Inform is not especially like Pascal or C, although probably closer to C than to anything else. It is an object-oriented like language, in that building blocks make up the structure of an Inform program; yet it is not "pure" in the way TADS is. Inform, while it has its own peculiarities, has its own elegance as well.

1. **Documentation, Weight 5. Rating: 4.** The Inform documentation has been the subject of recent newsgroup controversy. Graham has published three documents: *The Inform Designer's Guide*, *The Technical Manual and Z-Machine Specification*, and *The Craft of the Adventure.*

The Designer's Guide is the programming manual for the system. Esoterica are covered in the Technical Manual, while Craft of the Adventure deals with authorship and stylistic matters.

The second edition of the Designer's Guide is up-to-date with the current release of Inform, 5.5. The Guide is over 200 pages in length, with extensive discussion, examples, and exercises, with a good index. Nearly all of the omissions of the first edition have been cleared up and the Guide as it stands is complete and highly useful. It is available in all sorts of electronic formats, from an ASCII version suitable for on-line use, to a TEXversion which formats elegantly.

The Designer's Guide reflects Graham's eclectic tastes as well as his phenomenal literary knowledge and depth. As such, it is very much to the liking of some and less to the liking of others, on stylistic grounds. On technical grounds, some feel it is not as "beginner-friendly" as the printed TADS manual, and have criticized the exercises as being too difficult and requiring knowledge beyond the level the reader is likely to have reached at a given point.

Although I cannot be fully unbiased, having lived with Inform for a long time at an intimate level as the maintainer of the DOS ports of the compiler, I feel the criticisms are largely unwarranted. Separately written Inform tutorials are available, and the manual does not in fact purport to be a tutorial for beginners (as indeed, it is not). The manual *is* complete, well-written, and filled with useful examples. It is an indispensable reference guide which seldom fails the user who has gotten beyond the stage of rank beginner. And, for the beginner, a separately written hyper-text tutorial of high quality is readily available.

The Technical Manual will be needed by those few who want or need to do assembly-language Z-machine coding, or who are simply curious about the internals. The run time format is well documented and a wealth of detailed information is at hand.

The Craft of the Adventure is well-done and should be read by any prospective author regardless of the authoring system finally chosen. The paper gives some very good hints, suggestions, and ideas which are highly worthy of consideration.

2. **Ease of Use, Weight 5. Rating: 4.** Every language has its peculiarities, and Inform is no exception. Coding is quite clean and modular, with each object (be it location, actor, or material object) having its own logically-arranged block of code. Syntactical peculiarities creep in with the need to distinguish between "description" and "describe" and "name" and "parse_name", and various conventions about printing strings, return values, and other exotica.

Verbs are applied to nouns within the block referenced by the noun, similar to TADS. Instead of verification and action methods, Inform uses "before" and "after" methods, as well as an easy to use grammar table. Objects are easily reused; and the coding structure of even disparate objects is enough alike to make programming go a lot faster than it would otherwise.

It is my feeling, not backed by analytical evidence, that the neophyte will get more done at an earlier stage with Inform than with TADS; and at a later stage, when the language is learned well, there is little difference although TADS might have a small edge. This is not to say (like TADS) that Inform is a language for a neophyte. Again, with a powerful, broadly-applicable system such as Inform, it is only reasonable to expect the prospective author to spend time up front working with the language and learning to understand and leverage its many features.

3. **Parser, Weight 5. Rating: 5.** The parser is as good as they come, dealing well with compound sentences, antecedents, resolution of ambiguities and the normal range of parser problems and issues. In these respects it is the equal of the excellent TADS parser. The advantage of the Inform parser, though, is ease of modification and adaption, since it is largely contained in the libraries rather than in the program (as indeed, given the nature of the Z-machine, must be the case). Ease of parser modification comes up frequently in newsgroup discussions; surprising as this might seem to the beginning author, at a rather advanced level, this becomes an issue. It is also an issue, as noted once before, to writers wishing to create IF in a non-English language. (Off-the-wall irrelevant comment: some day I would love to see a parser for Tagalog. With its extensive use of passive voice constructions, it would be a challenge. Although, the Inform parser already has been translated into Spanish...)

4. **Support, Weight 4. Rating: 5.** Support for Inform is readily available on the newsgroup, and Graham, the author, is quite responsive to compiler-related questions, although he evidently receives all too much e-mail and should not be bothered unless there is no other alternative. Newsgroup support is generally very fast and of high quality. There are a few really high-level Inform experts who seem able and willing to respond to all sorts of enquiries about Inform programming.

   The various ports of Inform seem well-supported also. I jealously protect the integrity of the PC ports, and this approach is shared by the other porters as well— in fact, the Mac porter, Robert Pelak, is continuously enhancing the interface.

5. **Language Depth, Weight 4. Rating: 5.** Language depth is very great, nearly as great as TADS; and in fact it is possible to do just about anything in Inform that can be done in TADS, and in some cases a few

different things, which amounts to a lot of power. Inform offers a good set of operators, a library which provides a good set of objects and functions, and some unique and powerful features such as "scoping" rules. Scoping rules are a powerful and flexible means of determining when an object is in-view, "takeable", etc. (A significant part of the TADS "World Class" libraries deal with issues that amount to scoping questions.) Inform also allows for objects that span multiple locations, a useful feature for implementing something like carpet, grass, or snow.

Although this is perhaps more of an ease-of-use issue, I do feel that the advanced features of Inform are a bit less integrated than they are in TADS. Inform features have a layered feeling; they have been built-up as the language and especially the library have grown. You would expect this to be the case; we are dealing with the Z-machine and this imposes many restrictions. TADS is able to implement a lot more in the compiler and in the run-time, giving a better feeling of integration. (This is a mixed blessing, in that you can't directly alter the TADS compiler or run-time behavior.)

For instance, a limitation in Inform is on attributes and properties of an object. These attributes are a predefined set (the library leaves 11 of them free); and private global variables for an object are not available (you must use public properties). In TADS there are no restrictions. The Inform limitations arise mostly from the Z-machine, and largely cannot be avoided. An Inform author will often find the need to use a particular named attribute for some unrelated purpose; attributes may be reused and change their meaning; and all of this leads to the need to take extra care in coding.

But to return to the main point: the power of Inform is in the libraries, and the standard libraries are very powerful indeed.

6. **Portability, Weight 4. Rating: 5.** Inform games run on more platforms than any other system, period. The run-time, unlike the other systems, is *not* part of the Inform package. Various Infocom story-file interpreters are freely available and freely distributable. On the PC and Unix platforms, JZIP (John Holder's modification of Mark Howell's ZIP) is one of the best, and Frotz is a promising new entry in this field. Similar quality implementations exist for many other platforms, including some palmtops! It is safe to say that an Inform game is playable nearly anywhere...with one caveat.

   The new Z8 format Inform games, which allow for really large works such as *Jigsaw*, use a story file which can be as large as 512k. This, combined with the memory needs of the interpreter, can impose limitations on some machines, notably low-end PCs, with not enough available free memory.

However, any of the other formats, such as the common Z5, which still allows for a very large game, will run virtually anywhere.

7. **Run Speed, Weight 4. Rating: 4.** It is difficult to come up with an objective benchmark for interpreter run-time speed, and, with the variety of story file interpreters available, results are highly dependent upon individual environments. In addition, I have received quite contradictory reports from newsgroup correspondents.

   That said, my subjective impression has always been that Inform games run faster than TADS games, and I think measurement would bear this out. I think this has to do with the way TADS matches verbs to objects, but I invite commentary on this from the individual system experts.

   In any event, Inform games, even very large ones, have very good run-time performance on interpreters such as ZIP and JZIP.

8. **Literature Library, Weight 4. Rating: 4.** The Inform literature library has developed significantly in the past years. While the library of source code does not equal that for TADS, it is still large enough to provide numerous examples of how things are done. The *de rigeur* port of Colossal Cave is available, and smaller examples such as Toyshop and Balances were written to illustrate specific Inform features.

   The Inform literature library lacks large source code examples equal to *Legend* and such examples are unlikely to be forthcoming (although Graham Nelson is contemplating future release of the source to *Curses*). Nonetheless, there is more than enough presently available to be of great utility.

9. **Debugging Features, Weight 4. Rating: 3.** Inform has a few built-in debugging features, but they are not especially convenient and often more oriented toward debugging the compiler rather than debugging the game. The archive site has some "wizard" routines which help in debugging, and some of the interpreters have useful debugging features, such as watching the movement of objects.

   A source level debugger called "Infix" has been on the horizon for quite some time, and indeed, support for Infix debugging is built in to the Inform compiler. I have no further information about Infix, however.

10. **Future Prospects, Weight 3. Rating: 5.** Inform releases are reasonably frequent (from my alternate viewpoint as a port maintainer, less is better!) and generally of significant content. Graham Nelson retains a strong interest in his product. Version 6 is on the horizon as of this writing. All of this and the availability of all source code means that future prospects for Inform are very good indeed.

11. **Object Orientation, Weight 3. Rating: 4.** Inform calls itself an object-oriented language, and indeed it meets a loose definition of object-orientation. It is possible to define classes; multiple inheritance exists, as does overriding and a rather limited amount of data encapsulation (limited by the fixed property and attribute arrangement of the Z-machine). Methods can be arbitrarily complex and can be deployed in the expected places. Cross-calling methods from other objects can be done indirectly, although this is somethimes a nuisance.

    All of this lends Inform an object-oriented feel, as opposed to the more "strict" definition of object-orientation followed by TADS. I'd make a analogy here with the differences between C++ and Smalltalk. But for the author, the main features and advantages of object-orientation are fully available.

12. **Game Size, Weight 3. Rating: 4.** I'm rating this the same as TADS for different reasons. Recall that TADS can produce extremely large games at the expense of some run-time compatibility. Games produced by Inform probably cannot be as large as TADS games, yet they can be very, very large indeed in the Z8 format, and quite large enough in the Z5 format. The Z8 format also sacrifices some run-time compatibility (but considerably less so than TADS).

    What does this mean? Although I can't see a rating of "5" here for any system, until very large games are produced without impacting the run-time compatibility of a lot of platforms, still, the prospective author is not going to run into very many limitations here. A game such as *Legend* or *Jigsaw* can take years to produce. Most efforts won't push the limits of Inform's Z5 format, let alone require or be constrained by the Z8 format.

13. **Distribution Policy, Weight 3. Rating: 5.** Graham permits unrestricted freeware, shareware, or commercial distribution of games produced with Inform, asking only that a credit line appear in the game titles. Since free runtimes of high quality are available, distribution of Inform games is virtually without limitation.

14. **Programming Skill Required, Weight 3. Rating: 3.** As is the case with TADS, Inform is not the place to begin to learn how to program (although there are those few who have actually done just that). Familiarity with C would be extremely helpful; a background in Pascal would be the next best thing.

    The Inform manuals are not written at the beginning programmer level, either, instead implicitly contemplating a good familiarity with the basic concepts and constructs of a C-like language.

    Should the prospective new author without programming experience turn away from Inform (or TADS for that matter)? Not if the author is willing to take the time to become a reasonably good programmer. (If this is not the case, ALAN is an easier starting language.) Inform makes demands upon the author, as a powerful system will. Yet, once the learning curve is traversed, the rewards become well worth the effort. Inform is not easy; neither is TADS, C++, Smalltalk, playing the violin, or reading Aramaic. There is no free lunch or free ride; you have to put in your time up front.

15. **Cost, Weight 2. Rating: 5.** Inform is genuinely free.

16. **Source Code, Weight 2. Rating: 4.** All source code for Inform is readily available for every platform on which it runs (actually contained in a single version with multiple #ifdefs.) The code compiles on a wide variety of C compilers, from the humble Quick C to the powerful GNU gcc. However, I rate this category as 4 instead of 5 because possession of the source code does not mean you are likely to do a lot with it. I have spent a *lot* of time digging in this code, as the "generic" DOS port was at first pretty difficult; and I can say with complete assurance that the code is not to be trifled with. (This should not be surprising. Complex programs are often made up of complex code.)

17. **Compiler Speed, Weight 2. Rating: 3.** Inform compile times are noticeably slower than compile times for other authoring systems; previous informal benchmarks showed them to be consistently about twice that of TADS. I have not determined a reason for this.

    How serious an issue is this? On a slower platform, using the "generic" non-386 PC port of Inform, it could become an annoyance in developing a large game. Colossal Cave required five minutes or so to compile on my old (now gone) 186 portable. The same thing, of course, compiles in a couple of seconds on my Pentium 133 running Linux, so it's all relative. You will have to weigh the contemplated size of your game, your development platform, and your operating system and determine for yourself how important this may be. But on a "reasonable" modern machine, compilation in five seconds compared with compilation in ten seconds is not really an important matter.

    Again taking a reverse view, you might argue that slow compilation encourages you to code more carefully in the first place. You decide.

18. **Compiler Platforms, Weight 2. Rating: 4.** Inform has been ported to most all the popular platforms, and with availability of source code that is now rather clean, further ports should be no problem. Unix ports are trivial; the only platform with a doubtful future is the sub-386 PC platform. This port has been really difficult to keep up, and when Inform moved from release 5.4 to release 5.5, the libraries and code grew to a point at which the amount of free memory required crossed my "threshold" value

of 580k. Much beyond this point, we're starting to talk about special boot configurations and a fair amount of hassle to compile even small games.

I thought this was unimportant— who is still depending on a sub-386 PC?— and found that there are more prospective authors than I had expected who are using older equipment. So, for this reason, I assign a rating below the top level.

19. **Dynamic Object Creation, Weight 1. Rating: 3.** Well, it's not that you can *really* do dynamic object creation in Inform— but you can effectively fake it if you get deep enough into the advanced coding facilities of the language. But much more importantly, you can easily distinguish between many examples of a like object, such as snowflakes. So I'll go with a middle rating here.

**Summary Comment.** Inform has come a very long way and is solidly in the top tier of today's interactive fiction authoring systems. It has its own individual programming feel which is different from many other systems. It has an excellent library and a good set of facilities. Support is easily found and of top quality. While it still reflects limitations imposed by the Z-machine structure, it goes well beyond anything Infocom ever had for their own use. I think those folks would be pretty surprised at just how much Graham has done with their original ideas.

So, TADS or Inform? My answer is, of course, simply, "yes."

## 5.3 ALAN

**Release discussed: 2.6**
**Author contact information:** Thomas Nilsson, `thoni@softlab.se`

**Description.** I like ALAN (Adventure LANguage) a great deal. I've done some programming with it, and it is smooth. I admit I am biased, but I think for an author who is not a somewhat skilled programmer, ALAN is a fine choice, probably the best choice.

ALAN has a syntax which is very much declarative English (or Swedish) in format. There is an object/block structure which is easy to understand, and coding within each block is mostly uncomplicated. The ALAN runtime is pretty robust and not prone to mysterious crashes. Indeed, the authors describe ALAN as a "safe" language.

1. **Documentation, Weight 5. Rating: 4.** ALAN documentation is not as extensive as TADS or Inform documentation, but it's nice, complete, and quite suitable for a beginner. It is available in text form suitable for on-line usage, or in Postscript form which makes a very elegant printed manual if you can print Postscript. (And, the manual now prints properly on American "letter" paper as well as on European "A4" paper.) The

manual includes a brief but highly focused introduction to the art of writing a text adventure game. Every prospective author should read this as well as Graham Nelson's *Craft of the Adventure*.

The documentation could use more examples and sample code, however; and I did not find the BNF expression of ALAN syntax to be especially helpful (although perhaps this is due to my own ignorance). But as an example of readable, quality documentation, the ALAN manual is very good. I also congratulate the authors on their mastery of English, in their having produced something of a literary quality the typical native-speaking American couldn't come up with in a lifetime.

2. **Ease of Use, Weight 5. Rating: 4.** Let's be certain we understand what this rating means. If we are saying only, "is an authoring system easy to use" then ALAN would rate at the top of the charts. If on the other hand we say, and we do, "is it easy to use *and* is it easy to accomplish some particular reasonable task" then ALAN rates a bit lower. ALAN's simplicity provides ease of use; but when there is a complex task to accomplish which falls outside of ALAN's directly available structures, one must resort to various tricks and indirect implementations. The classic example is the use of the "description" sub-block to execute procedural code which cannot be easily executed elsewhere within an object.

   Nevertheless for straightforward operations ALAN is a snap to use. The learning curve is traversed quickly and results appear in short order.

3. **Parser, Weight 5. Rating: 4.** ALAN's parser is good, make no bones about it; it is nearly as good as TADS and INFORM, and correctly handles the expected range of conjunctions, prepositions, and so on, with the added benefit of handling English and Swedish (and I'd see little problem with Norwegian and not much problem with Danish). The drawback, and the reason for a slightly lower rating, is that it is fixed within the run-time module and there are no facilities at all for altering parser behavior (other than the "syntax" constructs which are part of the ALAN language).

   Now, I've been "selling" ALAN as a good first language for a new IF author. A new author really should be learning the basics and not be doing much parser modification. So, how much of a drawback this parser rigidity is, at the level of operation I'm contemplating, is uncertain.

4. **Support, Weight 4. Rating: 3.** Thomas Nilsson, one of the authors of ALAN, responds eagerly to questions about ALAN. Thomas is a complete gentleman, and provides excellent support for his product via e-mail. But unfortunately at present that is close to the only source of support. Only one expert, Mark Sachs, has appeared on the newsgroup, and Mark's availability and presence is uncertain. So, support for ALAN is not available through many avenues.

I sincerely hope that this situation changes as ALAN becomes better known and more widespread expertise develops.

In my opinion, a major inhibiting factor has been the ALAN compiler distribution policy. While the ALAN runtime and documentation are available from the archive site, the compiler is *exclusively* available by e-mail or snail mail directly from the authors. The ALAN terms of use specifically forbid placing the compiler on any BBS or Internet site. Thomas says this is because he wants to track the usage of ALAN, and while this is a reasonable wish, the ALAN compiler remains somewhat difficult to access. One must combine and uudecode a multi-part e-mail transmission, and this is hardly a task for a computing novice; and it is error prone as well.

5. **Language Depth, Weight 4. Rating: 3.** There are two classes of problems with ALAN's language depth. ALAN, by design, stresses a natural syntax and simplicity. This does not preclude a full set of features and a highly capable language; but it does limit how rich the feature set, ultimately, can be. I would not call ALAN's feature set sparse, but neither can it be called extensive and deep.

   While this requires rating this category lower than some other systems, it should not present a barrier to a prospective new author.

   The most serious lack, however, falls not in the design of ALAN but in the absence of a standard library. None is provided with the distribution, although a basic library, written by Mark Sachs, can be obtained from the archive site. Any user of ALAN is strongly advised to obtain this library. It is hoped that the library will be extended in the future.

   Actually, the sample games provided with ALAN show that a fair amount can be done without standard libraries, but a lot of default actions and results will have to be individually hand coded. This does reduce the appeal and utility of ALAN since the author ends up doing substantial extra work, all of which could be avoided with a standard library.

   It is my suggestion that ALAN enthusiasts join with ALAN's author to produce and document a good standard library.

6. **Portability, Weight 4. Rating: 3.** ALAN games are playable on most of the mainstream platforms: the PC, MAC, and Amiga are all covered, as are a few Unix platforms. A Linux port will appear very soon (I should know; I'm the porter).

   The version 2.6 PC compiler and runtime is limited to 386+ PCs, since they were built with djgpp. Thomas will later be producing a runtime for sub-386 PCs, at which time the portability rating should be increased by one level.

The one slight drawback, which is shared by TADS and many other systems, is that a platform specific runtime will need to be available to play an ALAN game. The ALAN runtimes can be rather large, especially on the PC platform.

7. **Run Speed, Weight 4. Rating: 4.** ALAN runtime speed is quite good, and the subjective feel is that it is comparable to any of the other faster-running major systems.

8. **Literature Library, Weight 4. Rating: 2.** There really is no literature library, and this is very unfortunate. A couple of small examples, which don't really show more than the basics, come with the ALAN distribution. Mark Sachs has produced a small but extremely useful set of examples, which he calls "etudes" and these should be obtained from the archive site. Mark's studies do illustrate how to use ALAN's more advanced features, and how to do more complex tasks with ALAN, information which is otherwise not available.

   One would hope that in the near future, at least Colossal Cave will be released in source form for the prospective ALAN author. And one would also hope that ALAN finally receives the attention it deserves and that ALAN games will begin to appear.

   Recently, a game (in runtime format) has been released as shareware for the PC, and other games are supposedly in progress; but source to these games is not available.

9. **Debugging Features, Weight 4. Rating: 4.** ALAN has some very good debugging features, ranging from internal tracing to dumping of tables. This forms a fairly complete debugging suite. The only lack is a source level debugger, which is a lack shared by nearly every other authoring system. But even better, the logical structure of an ALAN program encourages good, bug-free coding, making the need for a debugger less than it might otherwise be.

10. **Future Prospects, Weight 3. Rating: 3.** A minor release of ALAN has just come out after about a year since the previous release. Author Thomas Nilsson retains high interest in his product, but has a very busy schedule. He has projected a number of features for Version 3 but cannot commit to a target release date. Source code is not generally available for ALAN, and therefore future prospects for the language depend upon Thomas.

11. **Object Orientation, Weight 3. Rating: 3.** I'd say that ALAN has a "mild" object orientation; certainly there is no object orientation in the technical sense (although this is contemplated for a future release). Give ALAN credit for a nice block structure, prearranged objects such as

locations, etc. However, there are no notions of classes, inheritance, and the rest of the trappings of true object orientation.

12. **Game Size, Weight 3. Rating: 4.** I asked Thomas about size limitations within ALAN, since it wasn't possible to determine from the documentation just how large a game can be accommodated by this system. He responded that internal memory management was efficient and worked from a shared memory pool, so you couldn't really assign specific limitations such as number of locations permitted. "Available memory" is the ultimate limitation, and of course this will show up first in the sub-386 DOS implementation.

    This all means that ALAN should accommodate extremely large games, probably as large or larger than a Z8 Inform game, without hitting a limit except in the sub-386 DOS version. But, lacking concrete examples, I won't as yet go to the top rating in this category.

13. **Distribution Policy, Weight 3. Rating: 4.** Non-commercial distribution of ALAN games is not restricted, nor is non-commercial distribution of the ALAN run-times. I assume this applies to shareware. Commercial distribution needs to be arranged with the authors of ALAN.

    For non-commercial distribution, the authors do request, but not require, a source code copy of the game. I encourage authors who complete an ALAN game to honor this request and help build a library of ALAN code.

14. **Programming Skill Required, Weight 3. Rating: 4.** Of the language-like systems (in contrast to systems such as AGT or Gamescape) ALAN clearly requires the least amount of prior programming experience and skill. This is not to say that no prior experience is required, and hence the rating of 4 in this category.

    I continue to insist that there is no system within which non-programmers can be immediately productive. Experienced programmers nearly always underestimate the difficulties faced by non-programmers and think that certain languages or systems require "no" prior experience. It just doesn't work that way.

    In order to be successful with ALAN you must know how programs are structured, you must understand basic logical constructs, and you must be comfortable with a certain amount of syntax.

    But ALAN really goes a long way toward easing this task. For the almost neophyte, or even for the beginning IF author with a certain amount of programming experience, ALAN is an excellent first choice, perhaps the best first choice. The new author can program a small sample adventure and learn a great deal about the craft of IF; perhaps the author might even continue on with ALAN in the development of later, more sophisticated works.

15. **Cost, Weight 2. Rating: 5.** ALAN is a free system.

16. **Source Code, Weight 2. Rating: 1.** Source code is not available in general, although it has been made available to porters (I have it, for instance).

17. **Compiler Speed, Weight 2. Rating: 4.** The compiler is doing quite a bit of work in ensuring a "safe" run-time module, and does this work at surprising speed. While my statements here must be subjective, lacking a large test case to compile, ALAN compile speed is very good, and this speaks well of the internal design.

18. **Compiler Platforms, Weight 2. Rating: 3.** Compiler and run-time platforms are the same, and cover the mainstream including several Unix variations, with a Linux version available in the near future.

19. **Dynamic Object Creation, Weight 1. Rating: 2.** As is the case in many other systems, you cannot do this directly, but you can imitate the effect by moving objects in and out of the player's view. There does not appear to be a means of disambiguation of seemingly identical objects, although I have not experimented enough to see how this might be done, and I invite the reader to clarify this if possible.

**Summary Comment.** Program in ALAN for a few hours, as I've done, and it's difficult not to be charmed by the simplicity of the language, the ease with which the simpler tasks can be accomplished, and the overall pleasant feel of the system. Going a little deeper, though, one feels the lack of standard libraries and is somewhat constricted by the relatively small feature set. Still, the beginning IF author and ALAN seem a natural match. ALAN has received too little attention in the newsgroup and is too little known among IF authors. I sincerely hope that this situation will change before much longer.

## 5.4 Hugo

**Release discussed: v2.0**
**Author contact information:** Kent Tessman, `tessman@cibc.ca`

**Description.** Someone looking at Hugo who is familiar with other languages for interactive fiction authoring will quickly ask if this is Inform's junior sibling in disguise. In fact, it is not, and offers some surprising distinctions and extensions. Hugo may yet be a young and not fully developed language, but it has come a very long way in a very short time. Hugo shows great promise. Time will tell how much of this promise is realized- although I'm betting that the author, Kent Tessman, will carry Hugo some day to world-class status.

A look at a Hugo program will quickly show that the structure is clean, logical, and appealing. Hugo attempts to meld the straightforward approach of

ALAN with the power and depth of Inform. Of course, there are compromises and this unique fusion is partial rather than complete. But Hugo adds a few things all its own: a game-memory segmentation system which supposedly allows for very large games, and an actor scripting feature which is so logical and simple that one wonders why it isn't seen in any other authoring system.

1. **Documentation, Weight 5. Rating: 3.** Documentation is certainly complete and pretty easy to follow for the most part, although there are some really confusing sections. I'd advise the prospective Hugo game writer to browse code samples while reading the documentation. The manual does lack an index, and its rambling nature makes it harder than necessary to look things up. It would be well to keep the manual on-line to facilitate keyword searching, or to get busy with a yellow highlighter.

   I'm sure Hugo's author has been very busy with all of the work required to develop Hugo to its current state, but when a point of pause is reached, a reworking of the manual would really add a lot to Hugo's value.

2. **Ease of Use, Weight 5. Rating: 4.** Hugo's clean coding structure and style appears to lend itself to high ease of use. Most of the input I've received indicates that Hugo is somewhat easier to learn and use than Inform, even though Hugo has many Inform-like commands and statements.

   The primary factor usually cited here is a lack of "picky" syntax. Inform is very sensitive to misplaced commas and semicolons, and requires correct usage of both square and curly brackets. Hugo relies almost exclusively on curly brackets. You might or might not feel this is a more logical arrangement; certainly you need to take care with your visual formatting of your program. Look over the sample programs and draw your own conclusions.

3. **Parser, Weight 5. Rating: 3.** The Hugo parser is quite good enough, although I am told it is not presently as advanced as the Inform and TADS parsers. A significant advantage is that the usually modified sections of the parser are contained within Hugo's standard libraries (although the computationally expensive parts are in the runtime engine). It is also possible to override part of the parser's behavior with a pre-parsing routine, without the need to delve into the complexities of the entire beast.

4. **Support, Weight 4. Rating: 3.** I am sure this category will rate higher in the future, although for the present a few factors lead to a lower rating. First, newsgroup support for Hugo has only just now appeared, and indeed, there is not very much discussion of Hugo as yet. Also, the author of Hugo is quite busy and not always able to quickly respond to inquiries, leaving the Hugo game writer with nowhere to turn for immediate, expert assistance. A prospective Hugo author is, in any case, a bit of a pioneer at this point, and will have to call upon a good deal of self-reliance.

5. **Language Depth, Weight 4. Rating: 4.** I think some reviewers would rate this category lower, but Hugo provides all of the necessary language features as well as some unique extras. The manner in which objects and classes are treated (they are, or can be, equivalent) and the useful and straightforward implementation of inheritance lend a great deal of utility. The actor scripting feature is a real treasure for someone contemplating a game containing important non-player characters. Not only is it easy to script single NPCs, but NPC to NPC interaction is also possible. Even better, this scripting is contained within the standard libraries where it can be extended or customized.

6. **Portability, Weight 4. Rating: 3.** Hugo ports beyond the DOS platform have begun to appear. Earlier versions of the source were not ANSI-C compliant and made use of Microsoft specific extensions in the Quick-C language. The most recent release has cleaned this up a good deal; and Amiga and Linux ports have been completed. Other Unix ports should be easily achievable, and I imagine porting to a MAC, Acorn, or what-have-you should not be a problem.

   I expect that, by the next revision of this FAQ document, Hugo porting will be mostly resolved.

7. **Run Speed, Weight 4. Rating: 4.** Hugo run speed is comparable to any of the other well-running systems. There are no problems in this area; run speed is subjectively very good.

8. **Literature Library, Weight 4. Rating: 3.** This middle rating is based on the availability of the nearly obligatory port of Colossal Cave and a few other samples of code. The author of Hugo has published one substantial game in Hugo, and the source code has now been made available. There are also a couple of small demonstration games to be had. Beyond this, no literature library has yet developed. This would perhaps imply a lower rating, but the sample code available illustrates how to do things in Hugo pretty nicely, and the larger games are very useful.

9. **Debugging Features, Weight 4. Rating: 4.** Hugo has gone about the debugging chore by providing a special debugging library called HugoFix. This library contains a number of useful debugging features and commands, and should prove adequate for most debugging work. At present there is no source level debugger, and none is anticipated in the near future.

10. **Future Prospects, Weight 3. Rating: 4.** Based on history to date, I think Hugo has really excellent future prospects. Kent Tessman has come out with a steady series of new releases with significant improvements in them, and there is no doubting his serious interest in development of the

product. Full source code is available as well. The only reason for giving Hugo less than top rating in this category is its current status outside the mainstream of newsgroup interest.

11. **Object Orientation, Weight 3. Rating: 4.** While perhaps not meeting all the formal definitions of object orientation, Hugo's object nature is as appealing as it comes. Even though TADS meets a more strict formal definition, my own opinion is that Hugo's object implementation is really easy to use, giving the prospective author all the advantages of the object approach without the formality and syntax.

12. **Game Size, Weight 3. Rating: 3.** It was difficult to understand game size limitations based on the documentation and even the source code, so I asked the author for clarification. There is a hard limit of 32767 game elements, which is no limit at all. The author says he has cleaned up memory management, and he estimates that a game about four times the size of *Jigsaw* would be feasible in Hugo. However, he says that this would not run on an 8088 machine, and this is certainly true.

    I finally decided on a middle rating rather than one level higher since the existing DOS port of Hugo will reach the same limitations on any DOS machine, 8088 on up. The full capability of Hugo on 386 or higher machines could be realized should a DJGPP port of Hugo appear. Given the existence of a Linux port, such a DOS port should be trivial, but of course would exclude older DOS computers.

13. **Distribution Policy, Weight 3. Rating: 3.** My interpretation of the distribution policy as stated in Hugo's manual is that there is no restriction on freeware distribution of a Hugo game and the Hugo runtime. Shareware distribution, since it might involve transfer of funds at some point, appears to require prior arrangements with the author, and this is the cause of a slightly lower rating in this category. Commercial distribution clearly requires prior permission. Hugo's author has stated that he sees no problem, though, with shareware; he'd just like to know in advance.

    One other limitation the author has imposed is that modified libraries may not be distributed. This evidently refers to source modifications, since otherwise library hacking would be prohibited in any game that is intended for redistribution.

14. **Programming Skill Required, Weight 3. Rating: 4.** Hugo probably requires as much inherent programming skill and background as TADS or Inform, yet I think Hugo should rate higher because of its ease of learning. No, Hugo is not the place to begin learning to program, and the Hugo manual is hardly intended or suitable for non-programmers.

But the learning curve is somewhat less difficult to traverse. Still, prior background in C or perhaps Pascal is a necessity.

15. **Cost, Weight 2. Rating: 5.** Hugo is a free system.
16. **Source Code, Weight 2. Rating: 4.** Full source code is provided and the source code is not particularly difficult to work with, although the prospective source hacker shouldn't expect an easy go for a system this complex.
17. **Compiler Speed, Weight 2. Rating: 3.** Hugo, in the version I tested, is one of the slower compilers of the group, running about 50% faster than Inform (based on Colossal Cave). I did not test the pre-compiled headers feature of Hugo, however, and use of this feature would surely speed things up a great deal. I have had some correspondence which states that Hugo is supposed to be a fast compiler, but I haven't seen this as yet in practice. Hugo's author disputes my contention, saying that he has received a lot of correspondence saying how fast Hugo has become! I'll stick with a middle rating and let you experiment for yourself, and I welcome any sort of objective feedback which would help resolve this issue.
18. **Compiler Platforms, Weight 2. Rating: 2.** My comments on portability apply here. Hugo has been ported to some non-DOS/PC platforms, and moe ports will likely appear in the near future.
19. **Dynamic Object Creation, Weight 1. Rating: 4.** Hugo supports this feature in a reasonable manner; it especially supports dynamic object naming, a rather nice idea. In your game, you might want to create a situation where a player names an object, and can then refer to the object by that name. Hugo supports this easily.

**Summary Comment.** Hugo is going to be a player in the IF authoring arena, have no doubts about it. The portability and porting issues need to be solved, the manual could be refined, and the compiler could be faster. Hugo's author is dedicated to the success of Hugo, and progress has been very rapid. Should this continue, Hugo will enter the first tier before too long.

Hugo's combination of adequate features, including unique actor scripting, and a logical, appealing program structure, make it a unique system. It still needs some work, but check it out and see what you think.

## 5.5 AGT— Adventure Game Toolkit

**Release discussed: 1.83**
**Author contact information:** Mark Welch, `markwelch@trivalley.com`

**Description.** AGT, the Adventure Game Toolkit, is the professor emeritus of the interactive fiction world. In its heyday, there was nothing to equal it—

in fact, there was little else, and the aspiring IF author would simply choose it. Many, many games have been written with various versions of AGT, and some of these have been quite good.

AGT was, for many years, the shareware product of Mark Welch and David Malmberg; it grew out of Mark Welch's earlier effort called *GAGS: Generic Adventure Game System.* Then, as people do, David and Mark moved on to other things; AGT was released as freeware and the source made available.

Most of the mainstream editions of AGT rely on numbering systems; rooms, objects, messages, and words are all referred to by number. Certain limitations are hard coded into the system: the maximum number of rooms, etc., is fixed in advance, and allocation is not dynamic— you cannot trade room table space for object table space, and so on (although you can't do this in many other systems, either).

Games written with AGT can be of two types, "basic" and "professional." It is claimed that the basic games can be written without knowledge of programming, which is incorrect. Professional games are said to require some programming knowledge while offering, in return, some supposedly powerful constructs.

This all may sound a bit negative, and indeed, AGT does not meet today's highest standards for an authoring system. However, in its day AGT was clearly the best and this is not without reason. Although AGT really can't be recommended over TADS, Inform, and ALAN, it should not be cast aside.

And it hasn't been. As will be detailed below, some development is still proceeding which may give new life to this old favorite.

This discussion will concentrate on AGT version 1.83, updated by a third party from the last shareware release, 1.7. Version 1.7 was followed by another shareware release called *The AGT Master's Edition.* This latter release is considerably different from earlier releases, but as it concentrates new features in the areas of sound and graphics, it is somewhat out of the mainstream.

1. **Documentation, Weight 5. Rating: 5.** The excellence of the documentation now publicly available for AGT is beyond question. Suitable for beginner as well as expert, the documentation leads the prospective author step by step through the process by which an AGT game is written. An excellent language reference is provided, as are tips and techniques for designing a game and using AGT to bring it to fruition. Much of the Master's Edition manual, available in Word Perfect format, applies to the earlier editions as well. (I have had reports that the Master's Edition manual is not to be found. I do have an electronic copy of it and it should be generally available.)

   The ASCII text documentation can easily be used on-line.

   While the documentation is not as weighty as Inform's, or as well-written as TADS, it does appeal to the complete novice and should be read even if you don't plan to use AGT.

2. **Ease of Use, Weight 5. Rating: 3.** This category is hard to call. In most of the AGT editions, the use of numbers for everything is really a nuisance, and storing messages in a file separate from the code in which messages are used is very error prone.

   Some of these problems are solved in the Master's Edition; and for the other editions, some tools are available to help, such as a preprocessor which allows the use of alpha tags and converts them to numbers just prior to compilation.

   As a personal matter, I find the style of coding required by AGT to be a bit annoying. Actually, the style is pretty simple and straightforward, but is enough different from the procedural coding which I learned thirty years back to be an irritant. I suspect it would actually provide much less of an obstacle to someone with less of an ingrained bias.

   Some things, though, are indeed difficult to keep track of— "if-then-else" structures, for instance, require a clumsy structure to implement effectively, and, unless you fill your program with comments (which is always a good idea in any language), bugs are certain to creep in.

   Otherwise, though, AGT must be said to be easy to learn, and its greatest ease of use strength is that you can learn the features in layers. The "standard" set of commands and structures allows the prospective author to wet the feet and gain some experience, while the "professional" set of commands and macros provide a lot of additional power which can be learned and applied in increments.

3. **Parser, Weight 5. Rating: 4.** The AGT parser is criticized frequently in the newsgroup; indeed, there are some newsgroup correspondents who claim to be able to recognize an AGT game merely from the feel of the parser. I have always thought these criticisms and claims to be largely unfounded. In fact, the AGT parser understands and disambiguates a wide variety of English sentence structures. It is perhaps not as developed as the TADS or Inform parser, but it is very good nonetheless.

   Why, then, the ongoing criticism? I think the critics are confusing vocabulary limitations with parser limitations. The parser understands most of what is pitched at it within the limitations of the game's vocabulary. And unfortunately AGT does not have the capacity to deal with very much vocabulary: there is about a 500 word limit in the Classic Edition, and the Master's Edition only takes this to 1000 words. Too, many AGT game-writers didn't always make an effort to carefully define synonyms and alternative grammatical constructions in their games. (We must remember that these games were written at a time when the "state of the art" was less advanced.)

   The one grade lower rating of the parser has to do with its rigidity, as it is hard-coded into the runtime module. Since source code is available, it is possible to hack the parser, but this cannot be done conveniently via libraries or function calls.

4. **Support, Weight 4. Rating: 2.** I won't rate support at the bottom because undoubtedly limited support can still be found on the newsgroup. The authors, upon converting the system to freeware, have moved on and not surprisingly no longer provide product support. I believe, but have not verified, that some discussion of AGT takes place in one of the forums of one of the major on-line service providers.

   So, support is probably not zero, but it is not likely to be prompt or plentiful. There simply isn't enough current interest in AGT among enough people (although I do know a few enthusiasts are still out there).

5. **Language Depth, Weight 4. Rating: 3.** I'll go with a middle rating here, although let's say more of a low 3 than a high 3. You can do a lot of things with AGT; there are dozens of games out there which attest to this fact.

   The "standard level" of AGT is not very feature rich. You have locations, messages, a few flags to make use of, and a standard set of verbs. When you move to the "professional level" quite a bit more becomes available, although it is pretty rigid in structure, and it isn't really feasible to build new, complex commands out of the existing command set. You either like the AGT approach or you don't; there isn't much middle ground. For example, if you want to see if a container is open, you use the `IsOpen` predicate. This keeps the structure simple, but allows for no variation. If you want to test instead if an object is from an alien spaceship, you can't build an `IsAlien` macro; you have to use a flag variable instead to represent this idea.

   Similarly, the game structure itself is stratified. There are rooms, objects, monsters (similar to actors), and a few other categories; if you want something different, you have to substitute one of the pre-made categories instead.

   But to repeat, don't be misled. A lot has been done with AGT by clever authors, and, since there is a lot of source code around to use as examples, "unleashing" the inherent powers of AGT may be easier than doing the same on some other more advanced systems.

6. **Portability, Weight 4. Rating: 3.** I could perhaps have rated this category a little higher, but read on. Classic Edition AGT games are playable on the Atari ST, the Mac, and the PC. This covers probably 95 percent of the potential audience. In order to gain this portability, you need to distribute the correct runtime with your game files. Since AGT has gone through a number of iterations and versions, you need to make

available the exactly correct runtime for each platform on which you wish to release your game.

Other systems require the distribution of a runtime (TADS, Alan, etc.). But it's a bit easier for those systems, since the runtimes are more readily available and better coordinated.

The Master's Edition is limited to the PC platform.

7. **Run Speed, Weight 4. Rating: 4.** A few people think the AGT runtime is slow; actually it is quite good, certainly faster than TADS if perhaps slower than JZIP. The main objection is an overly-long time delay on the title screen which I have always found annoyingly reminiscent of shareware nag screens, which just beg to have a binary editor applied to them. But once in the game, performance is quite fine.

8. **Literature Library, Weight 4. Rating: 5.** Without a doubt there is more AGT source code available for browsing than for any other system currently available. Ranging from the obligatory port of Colossal Cave to a wide range of games of all sizes, there is plenty of code and a very high probability of finding an example of the sort of thing an author might be interested in coding.

   For a number of years, Softworks, the owners of AGT, sponsored an annual game writing contest. These contests drew a significant number of entries and helped to build a good sized library of AGT games. Recently, the source code for many of these games has been released, and this provides a very valuable resource for the prospective game writer. Something like thirty games are available in source form.

9. **Debugging Features, Weight 4. Rating: 3.** For debugging AGT relies on special "wizard" commands in the run-time, which allow you to track objects, jump to locations, etc. While certainly very useful, this does not tackle the question of program logic debugging in anything but an indirect manner. AGT's debugging features don't give you a means of looking at internal tables, such as can be done in Inform; and of course, there is no source level debugger available.

10. **Future Prospects, Weight 3. Rating: 2.** AGT isn't dead yet. There is the prospect of future releases by a third party— one of these has already appeared— and the source code has been published, although it's in Pascal, which is hardly a plus. A tiny bit of newsgroup discussion continues to take place. One can hardly rate AGT's prospects as very high, but they are at least non-zero.

11. **Object Orientation, Weight 3. Rating: 2.** AGT doesn't meet any sort of formal object oriented definition. I'll go a little above the bottom rating, however, since at least you code rooms, objects, etc., in blocks.

AGT's handling of verbs takes the reverse approach of TADS, Inform, and ALAN: a verb is coded and the objects and situations to which it applies are listed under the verb blocks. This is certainly not object oriented coding, but it's easy to see what goes with what.

12. **Game Size, Weight 3. Rating: 3.** There are some real limitations in AGT, particularly with respect to vocabulary size. Of course, the authors struggled with the need to make things run within the DOS memory limitations of the time (and indeed, which still are with us today). "Classic" AGT handles about 500 words. There are also limits of about 100 locations, a handful of interactive "questions" and so on.

    There is a variant of AGT called "AGTBIG" which increases these limits somewhat at the expense of more memory usage. The original packages were set up this way to accommodate some earlier PCs which didn't even have a full 640K of lower memory.

    Some medium-sized games have been produced with AGTBIG, and the "small" AGT is still capable of handling shorter "snack-sized" games, or better; after all, 100 locations is really quite a few. But AGT simply is not the vehicle for large or especially very large games.

13. **Distribution Policy, Weight 3. Rating: 5.** AGT is now freeware and there are no restrictions on game distribution whatsoever. (Even in its commercial days, game distribution policies were very liberal.)

14. **Programming Skill Required, Weight 3. Rating: 4.** AGT claims that "Standard" games require no programming skill. This is untrue. Standard-level games do not require knowledge of a formal procedural programming language; it isn't similar to some other languages, where you'd best know some C or Pascal before you begin.

    But, in writing an interactive fiction work, you are in every case, with every authoring system, writing a computer program of one type or another, and that is completely inescapable (unless someone takes your ideas and does it for you). To use AGT, you must understand at least something about logic structures, branching, testing, etc., even if it's called another name.

    AGT makes a valiant and somewhat successful effort to reduce the burden. But you can't make it go away. AGT requires only a limited amount of prior programming knowledge and experience, but that amount is non-zero.

15. **Cost, Weight 2. Rating: 5.** AGT is freeware.

16. **Source Code, Weight 2. Rating: 4.** Full source code to AGT was made available when the system became freeware. The source code, at least in my estimation, isn't at all difficult to read and work with. There

is one major exception, though, and the reason for a slightly lower than top rating: it is completely written in Pascal. I don't know about the availability of Pascal compilers on all supported platforms, but unless you have the right compiler, which will be a commercial product, the source code will do you little good.

Side note: I did attempt to compile AGT with the free GNU Pascal compiler, and about an hour of work produced no results. And, there have been comments on the newsgroup about using the GNU p2c tool to try to convert the system to a portable C format and make it work. This would surely breathe new life into the package.

17. **Compiler Speed, Weight 2. Rating: 4.** I had expected AGT to be a slow compiler, working from memories of experiences from years back. In fact, AGT proved a rather speedy compiler. Using Colossal Cave as a test case, AGT is noticeably faster than Hugo and Inform. The biggest delay seems to be in one or two "press enter to continue" screens which are really unnecessary.

18. **Compiler Platforms, Weight 2. Rating: 3.** Leaving aside the Master's Edition, the mainstream AGT variants work on the PC, the MAC, and the Atari ST. There is no version for any flavor of Unix or the Amiga, or any of the other computers out there. The omission of Unix is probably the most serious lack, yet it is understandable in a former shareware product targeted mostly to the DOS market.

19. **Dynamic Object Creation, Weight 1. Rating: 2.** As I do elsewhere, I don't rate this at the bottom because this sort of thing can be imitated. But true dynamic object creation and deletion, and disambiguation of multiple copies of a similar object, are simply not part of the system.

**Summary Comment.** AGT has a long history as a successful product and has the literature library to prove it. It has fallen out of favor as it has fallen behind. This does not mean it is a poor system or even necessarily a poor choice. It simply means that the newer systems have more capability.

But the last word may not be in. Recent uploads to the archive site have included this newest version of AGT, version 1.83, which has had extensions added by a current-day AGT enthusiast. More of this may come later; and if the aforementioned project to convert the system to portable C is ever brought to a successful conclusion, we may have to some day reevaluate the prospects of the AGT system.

## 5.6 Archetype

**Release discussed: 1.01**

**Author contact information:** Derek Jones, `dtj@rincon.primenet.com`

**Description.** Archetype is a modern example of an experiment in minimalist, object-oriented design. As an experiment, it succeeds well; well enough, in fact, to move out of the laboratory and into the field of practice.

Archetype programs have a unique look and feel. Like Hugo, that look and feel is one of simplicity and streamlining. Archetype is not as well developed, though, as some tier 2 languages, yet it is at a point where it can serve a useful purpose. Archetype assumes a certain amount of computer literacy, but with some background I believe the prospective author can start coding nearly at once.

1. **Documentation, Weight 5. Rating: 3.** Archetype's manual is complete, if spartan. It is available in both text and RTF formats, which is an interesting choice. Two documents are presented. One of them is a rather nice elementary tutorial on writing adventure games, with the bonus feature that it zeroes in on working with Archetype. The main Archetype manual is nicely laid out but rather brief. Everything necessary is covered but not in much detail; it is a minimalist manual for a minimalist system. The brevity of the manual is surely a reflection of author Derek Jones' lack of time; I would expect that the manual will develop in future releases.

2. **Ease of Use, Weight 5. Rating: 4.** Although I have not used Archetype beyond simple experimentation, I did not have any real difficulties in jumping right in, based on the manual and the coding examples present in the distribution package. Programming is pretty straightforward and nearly intuitive. Of course, we are talking about performing basic IF tasks; Archetype really doesn't as yet have facilities for very complex tasks (although these could be built up from the basic features). I have also received one direct report from a newsgroup correspondent who found Archetype especially easy to work with and extend.

3. **Parser, Weight 5. Rating: 3.** I'd rate the parser as average to adequate. My use of the sample games and review of the code did not find the parser to be spectacular, but neither could I say that it is not up to the task. A distinct advantage, of course, is that the parser is fully exposed; a disadvantage is that most of it is in the source code rather than in the libraries.

4. **Support, Weight 4. Rating: 2.** This rating is mostly based upon the fact that there is almost no newsgroup discussion of Archetype. I have had no input as to how support is from author Derek Jones, although he apparently takes an interest in his product as his schedule allows. But I think at present any kind of significant support would simply not be available.

5. **Language Depth, Weight 4. Rating: 3.** A minimalist experiment such as Archetype would not be expected to have great depth of features,

and Archetype does not. This does not mean the language is unsuitable for serious work, but it simply doesn't have the depth of Inform or TADS, or even Hugo. This same comment applies to the standard libraries, which are adequate but not more than that. Archetype in its present form is a framework rather than a fully-finished product.

6. **Portability, Weight 4. Rating: 2.** Why are so many things in the DOS world written in Pascal? Portability, unfortunately, has been seriously reduced by this choice of language, and until Archetype enters the mainstream of authoring systems, I can't see the likelihood of it being ported to other platforms. This limits an otherwise appealing language in a most unfortunate manner. The rating of "2" reflects DOS as a viable, widespread platform.

7. **Run Speed, Weight 4. Rating: 4.** Run-time speed is really very good. The sample games moved forward without any noticeable slowdown or difficulty.

8. **Literature Library, Weight 4. Rating: 2.** The literature library consists of two sample games. These are not at all bad, and illustrate the features of the system quite well. (There is as yet no Colossal Cave port.) Unfortunately there is as yet no literature library beyond this; hopefully this will develop in the future.

9. **Debugging Features, Weight 4. Rating: 3.** Debugging features consist of a small group of debugging directives which produce various quantities of output. This is no different than Inform or Hugo, although Inform and especially Hugo produce more, and more useful, output.

10. **Future Prospects, Weight 3. Rating: 2.** Perhaps this rating is lower than it ought to be, but I base it on several factors. There has been no release of Archetype beyond 1.01, and apparently the author, while retaining an interest, has only a little free time to devote to the effort. Other doubtful factors include the Pascal source code and complete absence of newsgroup discussion. Archetype has a fair potential but as of now, not much is happening.

11. **Object Orientation, Weight 3. Rating: 4.** Archetype by its very nature meets a high object orientation standard. Objects, object types, methods, etc., are all present and behave mostly as expected. The one lacking is the ability to pass parameters with messages. An object receiving a message knows where the message came from, but no more than this. In some situations this can be a handicap.

12. **Game Size, Weight 3. Rating: 3.** In the absence of any real information about game size, I'll go with a middle of the road rating. DOS

memory limitations will come into play at some point, but as with ALAN, I am not sure where that is. It is my subjective impression from browsing the code that a reasonably large game could be accommodated by Archetype.

13. **Distribution Policy, Weight 3. Rating: 4.** No statements are made anywhere in the distribution package about licensing or game distribution. I am assuming the typical arrangement that shareware distribution is acceptable and commercial distribution requires permission. However I have no facts or statements to base this on, and the author should clarify this important matter.

14. **Programming Skill Required, Weight 3. Rating: 3.** Prior coding experience and knowledge of a modern programming language is essential. Prior familiarity with object-oriented programming concepts is also highly recommended. While Archetype's reasonably clean structure and syntax do not put undue demands on the prospective author, and the learning curve does not appear to be very long, this is not something for the novice programmer.

15. **Cost, Weight 2. Rating: 5.** Archetype is free.

16. **Source Code, Weight 2. Rating: 4.** All source code is present in the distribution package. The only reason for a slight downgrade to the rating is that the source is Pascal, limiting its utility somewhat. My impression is that the code is pretty easy to read, and is well commented.

17. **Compiler Speed, Weight 2. Rating: 3.** Archetype's compiler isn't all that fast. Subjectively, based only on the 3000 line *Gorreven Papers* test case, it is faster than Inform and Hugo but slower than all the rest of the compilers.

18. **Compiler Platforms, Weight 2. Rating: 2.** My comments about portability apply here. The Pascal source code is a barrier to porting the system beyond its current DOS platform.

19. **Dynamic Object Creation, Weight 1. Rating: 5.** Archetype has full built-in features for dynamic object creation and destruction.

**Summary Comment.** Archetype is billed as a general system, with libraries supplied to gear it to interactive fiction. This appears to be true, and the generality of Archetype is greater than that of many other systems. The depth and development, though, is less, and it is hoped that the language will develop further in future releases. Archetype does have a sound, interesting underlying concept which embodies an easy to understand object-oriented approach combined with simplicity in coding style. Let's hope author Derek Jones has the time to work more with this system and bring it into the mainstream.

## 5.7 GINAS

**Release discussed: 0.4**
**Author contact information:** Jeff Standish, `jestandi@cs.indiana.edu`

**Description.** This little authoring system will warm the hearts of all Elisp programmers. (Elisp is the lisp-like programming language at the heart of the Emacs text editor.) Although still very much in the beta stage, with its fair share of bugs ("take all" causes a seg fault under Linux), the system promises to be, at the very least, of theoretical interest in the future.

The system appears to be in the nature of an elaborated experiment by the author in the creation of a Lisp-like interpreter tailored to interactive fiction authoring and play.

1. **Documentation, Weight 5. Rating: 3.** The manual formats to approximately 30 pages and is serviceable and adequate for most purposes, if at times a bit brief. There is enough content to allow you to dive into GINAS. The manual assumes familiarity with Lisp and object-oriented programming.

2. **Ease of Use, Weight 5. Rating: 3.** This rating represents an average which hides the truth. GINAS is probably impossible for someone not well versed in Lisp, but as I said above, could be the dream language for the experienced Lisp hacker. All of that aside, ease of use is somewhat limited by the small libraries, hard-wired parser, and the general need to "do it yourself" pretty often.

3. **Parser, Weight 5. Rating: 2.** It's not that the parser is particularly bad (or especially good) but it's buggy, and limited by the fact that it's hard-wired into the source code without the ability to override it in the game file. The author himself admits that this is one of the weaker points of the system. I suspect that future releases will show real improvement.

4. **Support, Weight 4. Rating: 3.** Although I have had little real contact with the author, the tone of the documentation indicates his genuine interest in ongoing improvements to the system, which would likely translate into adequate support. This category would rate higher except for the fact that there has been no newsgroup discussion of the system, and newsgroup support would be probably hard to come by.

5. **Language Depth, Weight 4. Rating: 3.** The language is not especially deep, although it supports a pretty reasonable, if small, subset of a typical Lisp language. This makes the language self-extensible if you have the time, talent, and inclination to develop it further.

6. **Portability, Weight 4. Rating: 3.** GINAS compiled with no fuss at all on my Linux system, which means that it will probably work on any

reasonable Unix system, and would work under DOS or Windows via the DJGPP compiler. The code looks quite clean and probably would port to sub-386 DOS machines as well; and similar prospects for other systems would be good.

Of course, none of these ports have been done, so a prospective author would have to compile and distribute the run-time with the finished piece, for each intended target system. This certainly lowers the convenience of distribution across multiple platforms.

7. **Run Speed, Weight 4. Rating: 4.** It is amazing that an interpreted language runs so well. Perhaps this is due to the small libraries and limited vocabulary presently implemented. But Colossal Cave, a supplied test case, ran quite nicely on my test system.

8. **Literature Library, Weight 4. Rating: 2.** The author has supplied the standard Colossal Cave port, some 4700 lines of code, and some other small examples, including Gold Skull from the TADS manual. But there is no other literature available— after all this is a new, experimental system.

9. **Debugging Features, Weight 4. Rating: 3.** The debugging features include the ability to run an interactive Lisp shell and a debugging sub-shell. In practice these again require a lot of familiarity with Lisp, and they are not especially well documented.

10. **Future Prospects, Weight 3. Rating: 3.** Perhaps I am being generous with this rating, but source code is available, and the author is actively pursuing improvements and fixes for a new release. Newsgroup interest has not yet developed, but nevertheless my instinctive feel is that GINAS does have at least some future prospects worth mentioning and watching for.

11. **Object Orientation, Weight 3. Rating: 5.** The system meets any reasonable definition for object-orientation. Class creation, inheritance, overriding, etc., are all supported. This would of course be expected in a language of this type, and adds greatly to its appeal.

12. **Game Size, Weight 3. Rating: 3.** A quick examination of the source code seems to indicate that game size limitations are probably controlled by available memory. This would allow for rather large games on a Unix systems and reasonable sized games under DOS. I've assigned this a middle rating in the absence of further information or experimentation, which perhaps a reader might be able to provide.

13. **Distribution Policy, Weight 3. Rating: 3.** The distribution policy for the interpreter and completed games is not stated. However, the

(uncompiled) package is available readily, and I would imagine that distribution of game files would not be a problem. Still, the author of GINAS should clarify this matter.

14. **Programming Skill Required, Weight 3. Rating: 2.** A great deal of programming skill and knowledge of Lisp is required, in my opinion, to work with GINAS. If you have it, GINAS will be a very interesting product. (As mentioned above, Emacs Lisp hackers will love it.) If you don't have the skill and background, GINAS is not for you.

15. **Cost, Weight 2. Rating: 5.** GINAS appears to be freeware.

16. **Source Code, Weight 2. Rating: 4.** All source code is available. The rating is 4 instead of 5 because it is not especially easy code to follow (although these things seldom are).

17. **Compiler Speed, Weight 2. Rating: 4.** Since there is no real compilation required in the present version, compilation speed and interpreter speed are equivalent.

18. **Compiler Platforms, Weight 2. Rating: 4.** I've just said that there is no real compilation, so why would "Compiler Platforms" rate higher than "Portability"? Simply because anyone with the requisite skills to use GINAS almost surely has the skill and ability to build an executable for any reasonable platform.

19. **Dynamic Object Creation, Weight 1. Rating: 4.** There should not be a problem in creating and destroying objects with the available Lisp interpreter. However I rate this below the top rating since, lacking some examples, my first look convinced me that there would be a lot of coding to do and that this would not be straightforward.

**Summary Comment.** There are some weaknesses in GINAS, such as the parser, a clumsy system for saving and restoring game states, bugs in the code, and a high skill requirement. All of these things make GINAS less desirable at present. But keep in mind that this is, after all, a beta version, and *is* an apparent experiment in Lisp-like authoring system design. GINAS has a lot of potential, even if it could not presently be recommended for most prospective authors. But I like the system for its innovative approach, its potential, and for its appeal to the inner hacker within me.

It might have been interesting had GINAS been implemented within the framework of Emacs lisp. This would have made some very powerful features available, although it would very severely limit the potential audience for the system.

## 5.8 LADS— Levi's Adventure Development System

**Release discussed: 2.1**
**Author contact information:** Peter F. Levy, 4209 Longmeadow Way, Forth Worth, TX 76133.

**Description.** This is a rather old system, which has survived and even been modestly improved upon. It started life on a TRS-80 and is now available in both source code suitable for use with the QBASIC or BASICA interpreters, and also in an executable format. The only viable platform is DOS at this point.

LADS seems deliberately designed to produce games with a distinct "Scott Adams" look and feel. Even the op codes seem much the same. Perhaps there is in fact a relationship of which I am unaware...

LADS is the minimalist approach to interactive fiction. The state of the art is today well beyond this, yet LADS is interesting both from the standpoint of nostalgia as well as being a workable, if primitive, system in its own right.

1. **Documentation, Weight 5. Rating: 3.** The documentation is bare-bones but quite complete and relatively easy to read. One nice feature about these docs is that you can read them over a couple of times in an hour, and then be ready to go to work on your piece. The docs would rate higher were it not for the extreme brevity.

2. **Ease of Use, Weight 5. Rating: 2.** Some might rate ease of use even lower. It depends on your background and what you are looking for. The use of numbers for everything, and the need to keep track of numbers for everything from objects to messages certainly lowers ease of use a great deal. The method of programming actions also requires a lot of careful planning, and actions of a complex nature are likely impossible. The rather cryptic action codes and instruction formats certainly add to the problem. No, this system is not very author-friendly.

3. **Parser, Weight 5. Rating: 2.** Parser format is rigid and built into the source code, and would be very difficult to change. The parser is, at least, of the three word type, which always must be verb-noun-object (which is really an indirect object, but the Scott Adams style grammatical lapse is preserved here). Vocabulary limits are also pretty severe (99 verbs and 99 nouns) so the player is likely to encounter guess-the-verb a little too often.

4. **Support, Weight 4. Rating: 2.** I'd rate this even lower, since there is little to no support on the newsgroup and I doubt if the author could even be located; but there actually are a few people interested in this language, with whom I've exchanged e-mail once or twice. Someone was interested enough to build an executable and clean up a few bugs, at least.

5. **Language Depth, Weight 4. Rating: 1.** Not to be insulting, but there is no language to have depth, just a small series of action codes which would not be especially easy to modify or extend.

6. **Portability, Weight 4. Rating: 2.** The only supported platform is DOS (I'm not counting the TRS-80). Ports to the Mac or Amiga should be possible. I made a feeble attempt to port it to Unix, using a BASIC interpreter available for Linux, and decided it would be more work than it would be worth.

7. **Run Speed, Weight 4. Rating: 4.** No surprise that the compiled version runs pretty well; it isn't really doing all that much and it ought to run quite fast.

8. **Literature Library, Weight 4. Rating: 1.** Well, there really is no literature in LADS that I have been able to find, although undoubtedly there are some games somewhere. The system comes with some small examples which are useful, but that's all there is.

9. **Debugging Features, Weight 4. Rating: 1.** There aren't any debugging features. You have to do it the old-fashioned way; sweat it out.

10. **Future Prospects, Weight 3. Rating: 1.** LADS isn't going anywhere. In the past year, LADS has been made to work on the PC platform, but that's all we're ever likely to see. In any case, why would anyone develop this system any further? There's no point in it.

11. **Object Orientation, Weight 3. Rating: 1.** In a language of this vintage, object orientation wasn't even heard of, let alone implemented.

12. **Game Size, Weight 3. Rating: 2.** A game with 99 rooms and 99 nouns, 99 verbs and effectively 99 objects (you can have 255 objects but only 99 can be referenced by the player) you might wish to rate this category even lower. Actually, though, the Scott Adams games fell well within these limits. You can write a decent mini-game within the size limits of LADS.

13. **Distribution Policy, Weight 3. Rating: 3.** In the absence of a formal policy statement by the author I automatically rate this category as middle of the road, but in fact I really couldn't envision any barriers to at least non-commercial distribution of works created with LADS.

14. **Programming Skill Required, Weight 3. Rating: 2.** How good are you with assembler language? You're dealing with op codes, operand fields, and all that stuff that "real, two-fisted" hackers eat for breakfast. If you're a mere mortal, this is not so easy. I'd rate it even lower except that the number of op-codes is manageably small.

15. **Cost, Weight 2. Rating: 5.** It's free.

16. **Source Code, Weight 2. Rating: 3.** Source code is freely available. I reduced the rating by two grades for the following reasons: the source code is in BASIC, and is difficult to work with. Programs of this vintage are not exactly works of art; in an effort to get them to function within the memory and interpreter limitations of the time, many compromises were made, and clarity, maintainability, and comprehensibility were the first things to go.

17. **Compiler Speed, Weight 2. Rating: 4.** The compiler (the executable version) is quite speedy, which again is understandable considering the relatively small size limitations on the game files.

18. **Compiler Platforms, Weight 2. Rating: 2.** The compiler runs under DOS; it might be portable to a Mac or Amiga; but ports to Unix appear problematical. Portability is not a strong suit for BASIC programs.

    Ideally, if there were enough interest and a reason to do so, a port to C would be the best answer. But I can't imagine why anyone would do this except as an exercise.

19. **Dynamic Object Creation, Weight 1. Rating: 1.** You can't do it, and you can't really fake it either. No surprise here.

**Summary Comment.**
Given the low ratings of this system, its lack of portability and difficulty of use, the low-end type of game it produces, and all the other limitations, you might wonder if you should bother with this system at all. In one unusual sense you would be wrong. Here's why.

A system of this type absolutely forces you to do advance planning for your game. You simply must map it out carefully, defining all the rooms, objects, and actions in advance. If you do not you will have no chance of ever coding it successfully. If you do lay it out well (and if you are comfortable with assembler-level coding), you have a very good chance of producing a completed game.

The conclusion I draw, then, is that using this system as an exercise in learning to plan a game in detail will serve the new author well when it comes time to plan and produce a larger, more serious work with one of the higher-tier authoring systems. (Of course, the flaw in this argument is that few prospective authors will want to deal with low-level coding.)

If you do nothing else, read the LADS manual some day, just to get an appreciation for how things were. And then realize that Scott Adams produced some interesting games with just such as system.

## 5.9 Gamescape

**Release discussed: A.4**

**Author contact information:** Dennis Drew, PO Box 101, Joplin, MO, 64802.

**Description.** No discussion of interactive fiction authoring systems is complete without a discussion of Gamescape, the system which bring us to the zenith of hyperbolic advertising. Gamescape is the creation of Dennis Drew, a professional programmer and a rather good comedian.

Gamescape has appeared in a number of versions, and it is the latest (and presumably last) version which is discussed here. This release made possible hi-res graphics as an option in the registered version.

Gamescape might possibly be called "AGT Lite" for it has a number of superficial similarities to the AGT system. Bu, it features a two-word parser, a strictly numerical coding scheme, and other trappings of a rather primitive and outdated authoring system, and falls far short of AGT in nearly all respects.

1. **Documentation, Weight 5. Rating: 3.** The shareware documentation is pretty good as far as it goes, but it doesn't go anywhere. Several sections are deleted and marked "registered only." Registration promises to reveal to you the "secret" and "powerful" commands which will allow you to do, it seems, great and glorious things.

   The documentation that is actually present is loaded with hype and advertising, but it does include a rather good short tutorial on how to construct an adventure game. This tutorial is worth reading no matter what authoring system you may use. Similarly, the explanations of how to use the unregistered version of Gamescape are well done and easy to follow, and have enough in-line examples to be really useful.

   I'd rate the documentation a little lower except for one factor. Nearly all of the "secret" commands are documented in earlier shareware versions of the product. If you are evaluating Gamescape it would be useful to look around for the earlier versions and study the more complete documentation which used to be provided. This earlier documentation explains "linking" of modules which is necessary to produce anything but a small game, and is now considered a "registered" feature.

2. **Ease of Use, Weight 5. Rating: 3.** Gamescape's ease of use is similar to AGT— although AGT has a structure which is easier to deal with. Locations and objects are all referred to by number, as are messages, and they all go in different sections of the input file. As with AGT, you'd better have an editor that allows you to work in multiple windows; otherwise you'll need some good notes on paper.

   One of the earlier versions of Gamescape had an input editor which helped you construct some of the locations and objects. This seems to have disappeared; perhaps it's moved to the "registered only" category.

   Otherwise, Gamescape should be no harder or easier to use than AGT. One unfortunate difference is that, while AGT would allow complete coding of a logic test ("is flag greater than 5" and things of that nature) on a single input line, Gamescape's primitive compiler can only handle one token per line. So, the aforementioned flag test would take three lines in the input file. This greatly impairs readability of Gamescape code and seems to be error-prone.

3. **Parser, Weight 5. Rating: 1.** The Gamescape parser is abysmal, understanding only subject and verb. There is no notion of adjectives or indirect objects. Adjectives must be attached to nouns with a hyphen, and the player must then type in the whole construct, such as "red-stone" or "small-box." Indirect objects are by implication only. If you need to distinguish between weapon categories in combat, for instance, you will have to resort to using signal flags and forcing unnatural actions upon the player (such as dropping all but the correct weapon for a given situation).

4. **Support, Weight 4. Rating: 1.** Perhaps this bottom-of-the-pile support rating is unfair, but let me explain the situation. First, newsgroup discussion is nonexistent— I've never seen a single posting dealing other than incidentally with Gamescape. Second, author support is only for registered users, and I don't know if that support is still available (I have no e-mail address for Dennis Drew). And, consider also a few statements in the documentation.

   We are told that if you are not a registered user, you will be "completely ignored" even if you submit a legitimate bug report. On the other hand, registered users are promised "full support" but with a warning about submitting bug reports. First, Mr. Drew informs us, that a system such as Gamescape which has produced "very complex" games, is unlikely to have any bugs in it. Second, he says, if you submit a bug report and it turns out that there was no real bug, but an error in system usage, he'll have to bill you for his wasted time. In other words, Mr. Drew does not seem to wish to hear very much about bugs in his bug-less system. (However, if you do find a real bug he promises to send you a "nifty adventure" in return.)

   Judge for yourself.

5. **Language Depth, Weight 4. Rating: 2.** There is little depth to the Gamescape system. You can do all of the typical Colossal Cave things, but there are really no advanced features, and static graphics in the registered version doesn't truly increase the usable feature set. Probably the best feature is that the system will deal with up to 1100 vocabulary words, which is a pretty good allotment for primitive authoring systems.

6. **Portability, Weight 4. Rating: 2.** Gamescape is a DOS only system. I rate portability above the lowest level not because there is any portability but because DOS covers probably 90 percent of the possible audience.

Also, Gamescape seems to be fairly simple in structure, relying on disk-based files rather than extensive use of memory, and will likely run on just about any DOS system.

7. **Run Speed, Weight 4. Rating: 4.** Run speed is quite fine. It ought to be; the run-time isn't really doing very much.

8. **Literature Library, Weight 4. Rating: 1.** A couple of small samples are provided with the system, and that is the entire extent of the Gamescape literature library. To my knowledge no other source code has ever been released for this system. In fact, I am aware of only two games written with Gamescape, both of them by author Dennis Drew.

9. **Debugging Features, Weight 4. Rating: 1.** There are no debugging features. You do it the hard way.

10. **Future Prospects, Weight 3. Rating: 1.** No source code is available, no newsgroup interest or even prospective author interest appears to exist, and it seems that even Dennis Drew has finally given up. No loss here.

11. **Object Orientation, Weight 3. Rating: 2.** Let's give Gamescape some credit for a certain amount of block structure, but as far as true object-oriented features, they are not here.

12. **Game Size, Weight 3. Rating: 3.** It is possible to write games that are arbitrarily large with Gamescape. But, in fact, there are some problems with this.

    First, the "linking" feature needed to build large games is now only documented in the registered version, although it was documented in earlier shareware releases and still seems to work in the same manner.

    Second, large games are written in segments. These segments cannot exceed 64k in size, which is fairly small. The segments must be "linked" together. Play in one segment can transit to play in another segment, and inventory and a few flags can be carried across. Other than this, there is no communication between segments, so messages, locations, objects, etc., that reside in one segment cannot be accessed by another segment. This is a very severe limitation. On the other hand, you can have as many segments as you wish— they are all disk-based— so, as long as your game can be written in more or less independent 64k units, the game can be as large as you wish.

13. **Distribution Policy, Weight 3. Rating: 2.** Gamescape's distribution policy has improved in the latest release, but it is still not very good.

    Freeware distribution of a Gamescape game is now permitted, along with distribution of the run-time engine, provided you don't use any of the "registered" features such as linking. This precludes distribution of any game over 64k in size. Shareware or commercial distribution is permissible only to registered users, in which case it is permitted without further restriction.

Dennis Drew imposes the ridiculous and probably unenforceable condition that, should you distribute a Gamescape game in violation of this policy, the game immediately becomes *his* property and you lose all rights to the game. I'm not a lawyer, so I suppose I shouldn't try to judge if a game is a derivative work and if derivative works can be confiscated, but the whole approach seems in tune with the way Mr. Drew attempts to do business.

14. **Programming Skill Required, Weight 3. Rating: 4.** I'll rate this the same as "standard-level" AGT, although AGT ease of use is greater. Again, familiarity with programming and program structures is needed, but prior experience with just about any language will serve. There is no need here to understand C, Pascal, or anything of that nature.

15. **Cost, Weight 2. Rating: 1.** The cost of Gamescape is $95 for registration plus $5 for shipping, or a total of $100, making this the most expensive system reviewed here by a factor of more than two. I wonder if *anyone* has ever registered this product. I certainly don't know what would possess them to do so.

16. **Source Code, Weight 2. Rating: 1.** No source code is available for this system.

17. **Compiler Speed, Weight 2. Rating: 3.** Compiler speed seems average; the sample cases are so small that a meaningful measurement really can't be obtained, however. But, given the simplicity of the system, I wouldn't expect any issues with compile speed.

18. **Compiler Platforms, Weight 2. Rating: 2.** Comments under portability apply here. The system is DOS only, but that does cover a large part of the market.

19. **Dynamic Object Creation, Weight 1. Rating: 2.** No real facility for dynamic object creation exists, however, this can be simulated by moving objects in and out of view.

**Summary Comment.** I include this review of Gamescape here only to show just how far things can go in the world of blatant commercialization, hype, and inadvertent comedy. I don't know why anyone would ever consider using Gamescape. It's primitive, limited in scope, and very expensive for what you get. Enough said.

## 5.10 Aventuro

**Release discussed: 2.1**
**Author contact information:** Wim Koolhoven, Rembrandtlaan 125, 7545 ZJ Enschede, Nederlando.

**Description.** There is probably no real reason to include a review of Aventuro in this document other than to recognize its unique character and its attempt to internationalize— perhaps, *de*nationalize— interactive fiction. Too, I am a language hobbyist— I've inflicted my brand of German on Volker Blasius a little too often— and this sort of thing fascinates me. So, bear with me as I use up ink/CRT space/bandwidth to expound a bit on this interesting authoring system. (Believe it or not, in 1994 my encounter with Aventuro was inspiration for me to buy some books and learn enough Esperanto to do at least simple reading.)

Aventuro was conceived to enable creation and play of interactive fiction in the Esperanto language. Esperanto is an "invented" language, created by Polish scholar Emile Zamenhof in the late 1800s. Esperantists propose it for an international language, and it is the only serious international language to have really taken hold (Interlingua is less known, and Klingon cannot be called a serious attempt at an international language, even though it now may be more widespread than Esperanto.) The appeal of Esperanto is its completely consistent grammar and pronunciation, and relatively simple structure.

If you cannot read basic Esperanto, Aventuro will be of no use to you. This will, I suspect, leave out 99.9 percent of interactive fiction authors and players, but still, here are my comments.

1. **Documentation, Weight 5. Rating: 2.** The documentation, all in Esperanto, of course, is sketchy and lacking in examples. The language is laid out in something that looks like BNF, and with enough study, a prospective author could probably put something together. But better documentation would be very helpful.

2. **Ease of Use, Weight 5. Rating: 3.** Ease of use for the author looks about average for a smaller system. A single input file, with a number of sections, comprises the game source file. Rooms, adjectives, direct objects, monsters, text strings, phenomena (actions), etc., are listed by number, section by section in the input file. The layout of each item in each section is quite logical, although some rigidity is present leading me to think that doing complex actions could be fairly difficult.

   A goodly amount of attention was given to allowing for proper expression of Esperanto grammar, such as correct gender and plural endings.

3. **Parser, Weight 5. Rating: 2.** The parser works, which is about all you can say. It accepts three word input, subject, verb, and object. It is

hard-wired into the code and there is no way to change it at all. If you need a flexible or adaptable parser, it's not here.

4. **Support, Weight 4. Rating: 1.** There is no e-mail address given for the author, the system was written in 1988 and the snail-mail address is likely invalid (although I haven't checked). There is no support on the newsgroup, although I suppose there is a slim chance of reaching someone who has used this system on the newsgroup `soc.culture.esperanto`.

5. **Language Depth, Weight 4. Rating: 2.** The language is a very basic one for interactive fiction design. A number of verbs and actions are hard-wired into the system, although making additions is quite easy. The system has a fixed directional movement system which does not allow for intermediate directions such as northeast or southwest. Objects may have numerous properties but these are all pre-defined, and there does not appear to be any means of changing these or adding to them. (It might be possible to do as is done in other languages, namely, using an otherwise unused attribute to represent something quite different from what the property name describes.)

   The language depth is adequate for smaller, less-complex works. But you aren't going to write *Jigsaw* or even *Trinity* with this system.

6. **Portability, Weight 4. Rating: 2.** There is no portability, period. I rate this 2 instead of 1 because the executables work on DOS systems, which comprise something like 90 percent of the potential audience. But beyond this, forget it. And the audience is limited to people with reasonable Esperanto fluency. (This assumes you write your game in Esperanto. There is nothing to stop you from using this system to write a game in English, but this would be a rather odd system choice in such a case.)

7. **Run Speed, Weight 4. Rating: 4.** No problems here. Again, a relatively simple system should not have execution speed difficulties.

8. **Literature Library, Weight 4. Rating: 1.** There is no source code literature available, and the sample data file which was supposed to be part of the distribution archive was missing, and I could not find it anywhere on the Internet or elsewhere. This is a really major handicap; given the terse documentation, examples are really necessary and there are none whatsoever. A compiled game, *La Insula Texel*, is included with the distribution, but there is no source code provided so as an author's example, it is not useful.

9. **Debugging Features, Weight 4. Rating: 1.** There are no debugging features. You do it the hard way.

10. **Future Prospects, Weight 3. Rating: 1.** No source code, no newsgroup interest, no literature available; written for Esperanto speakers only; no e-mail address for the author; what more need be said? Aventuro was a nice idea, but it's pretty much a dead system.

11. **Object Orientation, Weight 3. Rating: 2.** Although there is no implementation of an object-oriented paradigm, the manner in which things are grouped in the input file gives at least a little of an object-oriented flavor. There is limited cloning and inheritance in certain situations; see the section below on *Dynamic Object Creation.*

12. **Game Size, Weight 3. Rating: 3.** I could not readily determine the limitations on game size; the documentation is silent on this point (and on a lot of other points too). The games run memory-resident so the outer limit is of course free DOS memory. Internal limitations probably exist but were not tested. I solicit input from any reader of this FAQ willing to construct some tests.

13. **Distribution Policy, Weight 3. Rating: 4.** Non-commercial distribution of games and the run-time is explicitly permitted, and I imagine that, as is usually the case, this covers shareware as well. Commercial distribution specifically requires the written permission of the author. Good luck trying to find him.

14. **Programming Skill Required, Weight 3. Rating: 4.** An "average" amount of programming skill is required to use this system. It differs little in skill requirements from AGT and similar number and group oriented systems. This is again not to say that no programming skill is required. Such a system simply doesn't exist. But Aventuro doesn't require prior knowledge of C, Pascal, etc.; just familiarity with programming concepts.

15. **Cost, Weight 2. Rating: 5.** It's free.

16. **Source Code, Weight 2. Rating: 1.** Completely unavailable; might not even exist any longer!

17. **Compiler Speed, Weight 2. Rating: 4.** Well, honestly, I didn't test this, since there are no test cases and I didn't want to take the time to develop one. Any interested reader is invited to help out. But from the looks of the language I don't think compilation speed can be an issue.

18. **Compiler Platforms, Weight 2. Rating: 1.** It runs under DOS, period, with no possibility of creating other versions.

19. **Dynamic Object Creation, Weight 1. Rating: 3.** Here is something really amazing: the phenomena choices (hard-wired actions) include creating new adjectives and monsters, including making modified copies

of existing ones. Of course, these have to be defined in advance, so in one sense it isn't truly dynamic object creation; nevertheless, this is an unexpected feature.

**Summary Comment.** Even putting aside the question of fluency in Esperanto, this would not be a system which would be a reasonable choice for a prospective author. There is simply not enough documentation and no examples to guide the way. And, returning to the language question, if you did manage to complete a game, who would play it? Perhaps Esperantists would all play it merely because it exists, but there would be little other opportunity to have your work enjoyed by the public.

# 6 Final Remarks

In choosing an authoring system, it is unlikely that one will err in choosing either of the Tier 1 systems, TADS and Inform. Choosing between them comes down to determining which individual criteria are important to you, your writing and programming style, and the particular work you might have in mind.

Perhaps you believe strongly in the free software philosophy, or wish to undertake a commercial project without seeking further licensing. Inform might then become your choice. Or, perhaps strong object orientation, sheer speed, and a choice of libraries may be your main concern, and in such an instance you may choose TADS.

But it's quite clear, and `rec.arts.int-fiction` postings seem to show without much doubt, that you will not go wrong with either of the Tier 1 systems. It's very much a matter of choosing one, staying with it until you are comfortable and learn it well enough, and carrying through to produce your work.

Still, it is not out of the question to choose one of the better Tier 2 or Unprocessed Tier systems. ALAN, a traditionally under-appreciated entry, offers ease of learning and a kind of classic elegance at the expense of richness of features; but for someone with a less technical programming background, or for projects which are more straightforward, ALAN could indeed be a fine choice. One might wish, in fact, that it were used more than it is today. AGT, although older and with more limitations, has been the vehicle for countless excellent IF works, and would still serve today for many projects. Hugo is an up-and-coming language and deserves watching, and, perhaps, a good look to see if it meets your needs. Archetype also has possibilities and one can hope that the author finds time to develop it further.

Choosing a Tier 2 or Unprocessed Tier system does, of course, mean that the prospective author is assuming some increased risk, particularly in the areas of run-time compatibility and feature sets. Whether that risk is sufficiently offset by ease of use or some other criterion again becomes a personal choice.

Choosing a Tier 3 system is not recommended for most authors or projects. Yet, it is not out of the question on an experimental basis or for some special reason. GINAS provides a fascinating framework and it might be interesting to attempt a piece with it just for fun. LADS, while a primitive system by any standard, produces games which have a "Scott Adams" feel, and it might be interesting to try it as an exercise in minimalism. But, beyond this, most of the Tier 3 systems are simply too limiting in too many dimensions to be a logical choice over a Tier 1 or Tier 2 system. In a few cases, the systems are downright bad news.

So, if you've read this far expecting to have the choice of authoring system made for you, you're at the moment of disillusionment. You won't go wrong in Tier 1, and you may find something that fits you and your project in Tier 2 or the Unprocessed Tier. And now, the rest is up to you.

# 7 Afterword: How Many Systems Are Enough?

In this paper we've looked at a considerable number of interactive fiction authoring systems, and we certainly haven't covered all of them. There are easily ten more publicly available systems, and there are probably just as many proprietary systems which were developed at one time or another. Infocom had their own Lisp-like language, *ZIL*; and the trivia award goes to anyone who can place the *King Edward Adventure Language.* The list goes on.

The last year or two has seen the appearance of *Hugo, Archetype,* and *GINAS*. Another language, *Snack,* is reportedly in preparation. Hardly a month goes by without a newsgroup posting which starts, "I'm writing a new adventure game language, what would you like to see in it...?". Are we reaching the point of too much of a good thing? Is enough indeed enough? How much attention will the new languages get? Are we fragmenting the IF author community?

The current "marketplace" has two front-runners, Inform and TADS. These are so far ahead of the rest of the pack in terms of usage and attention that there is little contest. Yet, Hugo promises to deliver and ALAN has its place. And so on for a few of the others.

I can't help but feel that if someone wants to develop a new language which achieves some special purpose, even if purely experimental, or deals with some perceived lacking of the other authoring systems, then they should be encouraged to do so. In the long run I think the generation of new ideas benefits the IF community, even if a new language is never used by someone other than the author. GINAS, for instance, isn't going to be a mainstream choice. Yet, the very existence of GINAS makes available another view into the problem, a view decidedly different from many of the others.

There is, though, a most relevant caution raised by one correspondent: you should be familiar with what has already been done if you wish to do meaningful new things.

Interactive fiction is an appealing blend of art and technology. For those to whom it appeals, there can never be "too much" of any of it.

# 8 Interactive Fiction Resources

I'll not try to list the numerous Web pages which are now accessible. Instead, I'll just mention the major Usenet newsgroup, `rec.arts.int-fiction`, which is dedicated to interactive fiction authoring issues. (Questions about actual game play belong on `rec.games.int-fiction`). Electronic magazines about IF include Eileen Mullin's excellent *XYZZY News*, copies of which can be found on the archive site, and Whizzard's *SPAG*, which concentrates on game reviews and can also be found on the archive site.

All of the authoring systems listed here and a wealth of other material, including games, articles, and newsgroup archives, can be found at the interactive fiction archive maintained by Our Hero in Sankt Augustin, Germany, Volker Blasius. Access is via anonymous FTP to `ftp.gmd.de`, subdirectory `if-archive`. Volker has done us all a tremendous service in organizing this archive and keeping it up to date.

Finally, I can't pass by the opportunity to give a plug for my own IF resource on my dial-up bulletin board system, GobblerNet Adventure Games. Located in Bismarck, North Dakota, the home of the eternal frost, GobblerNet has on-line most of the resources of the Internet archive site, and many other materials including a very large `ZZT` collection. GobblerNet can be reached at 701-222-0429. This is, of course, a long-distance call for just about everyone!

# 9 Disclaimer

The author of this FAQ document has made a good faith effort to provide accurate information, but assumes no responsibility of any kind for any damages of any type arising out of errors or omissions in the document, nor for any damages of any kind arising from the usage of this information in any manner whatsoever. You must evaluate this information for your own purposes, and you assume all risks and liabilities in conjunction with its use. The author may not be held liable for direct, indirect, special, consequential, or any damages of any type under any legal theory whatsoever, even if the author has been made aware of the possibility of such damages. If your house collapses, get it fixed. If your dog dies, give thanks, and cook it and eat it. If fluctuations in the geomagnetic field disturb your rest, consult your physician or psychiatrist.

# 10 To-Do List

Wishing to release this first version of this FAQ, some things were left undone, both large and small. I hope to include as many as possible in future versions (expected to be about twice annually).

- Another rating category should be added which deals with the "professional appearance" of a game produced by a certain system. Does the screen format nicely? Are there special features, such as bolding of text? How is the display organized?
- Reviews of other, older systems such as ADL, AdvSys, etc., should be included. I'll be adding one or two of these at each new FAQ revision.
- Reviews are needed of newer systems for platforms to which I don't have easy access, such as Rexx Adventure for OS/2, etc.
- An effort is needed to make the document less PC-centric. I've been told, quite correctly, I fear, that my view is a bit myopic here.
- I'd like to publish objective measurements of compiler speed, runtime speed, and game size limitations.
- I'd really like more input from real experts for the AGT review.
- ...and so much more!

# 11 Comments on this FAQ

Comments, corrections, additions, suggestions, and constructive criticism on this document are more than welcome, earnestly solicited, and will be gratefully received. It is only with your help that the document can be improved and made more useful. Please email `bnewell@gobblernet.sputnix.com`. If this doesn't work please try `bnewell@btigate.com`, `bnewell@delphi.com` (which may soon disappear), or post to `rec.arts.int-fiction`.

Snail mail, if you really want to send it, goes to:

```
1825 North Grandview Lane
Bismarck, North Dakota
USA 58501
```

Please include some sort of electronic contact address, as answering snail mail is likely to be glacially slow, if I even get to it!

I am always interested in having coffee, a beer, a glass of wine, or even lunch or dinner, with other newsgroup correspondents. I've met a few of you, and hope to someday meet more of you when the opportunity presents itself.

And now, choose your system, write your game, and publish it. We're all waiting, you know.

## 12 Credits

Thanks for assistance of all sorts with this FAQ go to many people, but especially to Jools Arnold, who suggested I expand my old FAQ; Kent Tessmann, Cardinal Teulbachs, Thomas Nilsson, Graham Nelson, the correspondents on `rec.arts.int-fiction`, and Volker Blasius for all-around inspiration (and also for being the only fellow on the newsgroup older than I am!).

## 13 Revision Log

**0.1** February 20, 1996. Initial publication.

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TADS | 4 | 4 | 4 | 5 | 5 | 4 | 3 | 5 | 4 | 3 | 5 | 4 | 4 | 3 | 4 | 2 | 5 | 4 | 5 |
| Inform | 4 | 4 | 5 | 5 | 5 | 5 | 4 | 4 | 3 | 5 | 4 | 4 | 5 | 3 | 5 | 4 | 3 | 4 | 3 |
| ALAN | 4 | 4 | 4 | 3 | 3 | 3 | 4 | 2 | 4 | 3 | 3 | 4 | 4 | 4 | 5 | 1 | 4 | 4 | 2 |
| Hugo | 3 | 4 | 3 | 3 | 4 | 2 | 4 | 3 | 4 | 4 | 4 | 3 | 3 | 4 | 5 | 4 | 3 | 2 | 4 |
| AGT | 5 | 3 | 4 | 2 | 3 | 3 | 4 | 5 | 3 | 2 | 2 | 3 | 5 | 4 | 5 | 4 | 4 | 3 | 2 |
| Archetype | 3 | 4 | 3 | 2 | 3 | 2 | 4 | 2 | 3 | 2 | 4 | 3 | 4 | 3 | 5 | 4 | 3 | 2 | 5 |
| GINAS | 3 | 3 | 2 | 3 | 3 | 3 | 4 | 2 | 3 | 3 | 5 | 3 | 3 | 2 | 5 | 4 | 4 | 4 | 4 |
| LADS | 3 | 2 | 2 | 2 | 1 | 2 | 4 | 1 | 1 | 1 | 1 | 2 | 3 | 2 | 5 | 3 | 4 | 2 | 1 |
| Gamescape | 3 | 3 | 1 | 1 | 2 | 2 | 4 | 1 | 1 | 1 | 2 | 3 | 2 | 4 | 1 | 1 | 3 | 2 | 2 |
| Aventuro | 2 | 3 | 2 | 1 | 2 | 2 | 4 | 1 | 1 | 1 | 2 | 3 | 4 | 4 | 5 | 1 | 4 | 1 | 3 |

Table 1: Ratings by Category Matrix

## 14 Summary Table

Evaluation categories by number:

1 Documentation
2 Ease of Use
3 Parser
4 Support
5 Language Depth
6 Portability
7 Run Speed
8 Literature Library
9 Debugging Features
10 Future Prospects
11 Object Orientation
12 Game Size
13 Distribution Policy
14 Programming Skill Required
15 Cost
16 Source Code
17 Compiler Speed
18 Compiler Platforms
19 Dynamic Item Creation