Inform 7 and the Teaching of Writing

Brendan Desilets

Can students improve their writing and thinking by programming computers? In his seminal book *Mindstorms: Children, Computers, and Powerful Ideas (1980),* Seymour Papert argues that, within a computer-rich environment, or "microworld," students can construct knowledge in uniquely exciting ways, largely through the use of the user-friendly programming language called Logo. Taking their cue from Papert, teachers of writing have developed ways to apply Logo to the writing process, even in conventional instructional settings that do not offer a great deal of computer access. In one instance, teachers have used the recursive power of Logo to help students develop their ideas through extended definition (Desilets 43).

Still, Logo has its obvious drawbacks as a tool for teaching writing. To begin with, Logo programs, or "source text," don't have much in common with actual essays and stories. For example, two Logo programs, or "procedures," which can be used in teaching students to develop essays through extended definition, look like this:

```
TO SQUARE :SIZE
    FD :SIZE
    RT 90
    FD :SIZE
    RT 90
    FD :SIZE
    RT 90
    FD :SIZE
    RT 90
    END

TO GROWSQUARES :SIZE
    SQUARE :SIZE
    RT 20
    GROWSQUARES :SIZE + 5
    END
```

Beyond Logo: What Would Writing Teachers Want?

This Logo code is easy enough to teach, and it includes some "powerful ideas," such as the use of a variable (:SIZE) and the odd notion of recursion, through which the procedure called "GROWSQUARES" starts increasingly large iterations of itself (GROWSQUARES :SIZE +5). However, if we want to instruct students the writing process without having to teach for transfer in a

very vigorous, time-consuming way, we would be better off with a programming language whose code looks more like an essay. In addition, Logo code generally does not produce output that looks like an essay or narrative. In the example we've reproduced above, the output is an ever-increasing spiral drawn on the computer screen—an admirable way to show what recursion is, but a long way from a written text. For teaching purposes, most of us would prefer a programming language that can produce an interesting prose output.

Is there a programming language that uses essay-like source code to produce useful prose output? Yes, but it hasn't been around for long, only since 2006. The language is called Inform 7, and it's used to create computer-based works of literature called interactive fictions. Before we consider the virtues and limitations of Inform 7 as a tool for teaching writing, we'll need an understanding of this new form of literature.

Interactive Fiction

Interactive fiction, sometimes called text adventure gaming or IF, is a form of computer-based literature in which the reader (or "interactor") controls the actions of the main character by typing ordinary English sentences at a computer keyboard. It's a bit like a video game, but in words. IF began in the late 1970's, experienced a commercial heyday in the 1980's, became a focus of talented amateurs in the 1990's, and is enjoying vastly renewed interest in the last four years, with meetings of enthusiasts around the country, two new commercial projects, and even a documentary film (Scott). Authoring systems for interactive fiction have been available to the general public since 1987, but Inform 7, which debuted in 2006, is the first to allow the writer to compose source text in ordinary English sentences.

Each work of interactive fiction, like other forms of fiction, operates within an artificial "model world," but, in interactive fiction, the reader can manipulate this little world by typing sentences at a computer keyboard. In order to interpret the English sentences that the reader types, a work of interactive fiction employs a "parser," a programming device that translates the user's prose input into a form that the story can respond to (Montfort 107).

Here is a transcript of a session with an extremely (perhaps ridiculously) simple instance interactive fiction. The simplicity of this example story will prove useful when we look at the sort of writing students might do with Inform. The reader's input, which would vary greatly from session to session, appears in boldface type.

```
It's another typical day in College Writing II.  The professor is
illustrating the use of thesis statements, using a whiteboard, when
she drops her marker on the floor.  She appears to be a little
hesitant to bend down and pick it up.

A Narrative About Virtue
An Interactive Fiction by Brendan Desilets
Release 1 / Serial number 101106 / Inform 7 build 6E72 (I6/v6.31 lib
6/12N)

Coburn 301
A typical college classroom, with lots of desks, a whiteboard, and a
dozen or so students.
```

You can see the professor, a marker, and Matt here.

>Look at the desks.
Ordinary desk-chair furniture.

>Look at the whiteboard.
An ordinary whiteboard, which the professor has been using in her presentation.

>Look at the professor.
A smart and kindly lady, but elderly and not too spry.

>Pick up the marker.
As you pick up the marker, you notice that there's a Zune MP3 player on the floor, under your chair. The Zune isn't yours, though you wish it were. In fact, you recognize the Zune as the property of Matt, who's slouched beside you.

>Take the Zune.
Taken.

>Give the marker to the professor.
The professor thanks you with notable sincerity. You have demonstrated the virtue of thoughtfulness.

[Your score has just gone up by five points.]

>Look at the professor.
A smart and kindly lady, but elderly and not too spry. The professor looks pleased with you.

>Give the Zune to Matt.
Matt nods. He looks a little surprised at getting his toy back so easily. You have demonstrated the virtue of honesty.

The professor turns to you and asks, "Now, Jamie, can you tell us who wrote Plato's *Republic*?" Unfortunately, you've never heard of this work.

The professor says, "Jamie, I know you can figure this one out. Who wrote Plato's *Republic*?"

[Your score has just gone up by five points.]

>Look at Matt.
A sincere, if slightly lethargic young man, wearing a UMass Lowell sweatshirt. Matt looks quite happy right now.

The professor says, "Jamie, I know you can figure this one out. Who

wrote Plato's *Republic*?"

>Say Aristotle to the professor.
The professor says, "No. That's not it."

The professor says, "Jamie, I know you can figure this one out. Who wrote Plato's *Republic*?"

>Say Plato to the professor.
The professor expresses appreciation of your insightful response. You have demonstrated the virtue of intelligence.

*** You have won ***

You scored 15 out of a possible 15, in 13 turns.

(An interactive version of this very brief story is available on the Web, using this link. Some examples of real, fully developed interactive fiction are at http://pr-if.org/play/.)

Inform 7

How does Inform 7 enable a writer to produce an interactive story? Like all IF authoring systems, Inform 7 provides a parser that can accept a reader's input. In addition, Inform offers ways for the author to manipulate the parser for the purposes of a particular story. Further, Inform offers a model world that the writer can use, altering it as he or she will. This model world includes settings (called "rooms"), doors, items that can or can't be carried, containers, locks and keys, and characters. The great innovation that Inform 7 offers to IF authors is the ability to implement all of the above, while staying, pretty much, within the parameters of readable English prose, with typical sentences, punctuation, and paragraphs (Nelson).

Generally, when an author produces a work of interactive fiction, he or she writes "source code," a set of instructions for the computer to use in presenting the story to a reader. The writer then tells the authoring system to transform the source code into a form that a computer can understand. This transformation is called "compiling." The Inform 7 source code for an interactive story takes the form of a recognizable essay developed through process analysis. Using English sentences, the source code tells the computer, step by step, how to present the story to the reader.

The process of compiling source text offers some useful opportunities for the teaching of writing. In order to compile source code, the Inform system has to check the code for a variety of features, including spelling, punctuation, vocabulary, transitions, and completeness. In other words, the compiler checks for textual features that relate directly to what writing teachers call clarity, coherence, and development. If the source code fails to live up to the compiler's expectations, Inform 7 will fail to produce a work of interactive fiction, issuing an error message. In other words, Inform 7's compiler will act somewhat like a very strict teacher of writing.

A Look at Inform 7 Code

Let's have a look at some source code, to see how these ideas work out in practice. Once we have set up an Inform 7 project, by specifying its title and author, the system offers us a nearly blank pane in which to place our code. The pane looks like this:

```
"A Narrative About Virtue" by Brendan Desilets
```

In this pane, let's create our first room, using English sentences.

```
"A Narrative About Virtue" by Brendan Desilets

Coburn 301  is a room.  The description of Coburn 301 is "A typical
college classroom, with lots of desks, a whiteboard, and a dozen or
so students."
```

At this point, if we tell Inform to compile our code, Inform will do so, placing our player/character of the story in a room called “Coburn 301” and described as “A typical college classroom, with lots of desks, a whiteboard, and a dozen or so students.” The reader won't be able to do much of anything yet, but the story will, at least, start. Suppose, though, that we misspell “description.” In this case the compiler will refuse the create a story, complaining as follows:

```
Problem. You wrote 'The desciption of Coburn 301 is "A typical
college classroom, with [...] board, and a dozen or so students."'
but this seems to say that a thing is a value, like saying 'the chair
is 10'.
```

The compiler thus has a role in enforcing clarity by insisting, up to a point, on correct spelling. Similarly, it will insist on a controlled vocabulary (“room”) and completed punctuation, such as the closing quotation marks. What's within the quotation marks, on the other hand, is, for the most part, none of Inform's business. The compiler will not complain if a student misspells “whiteboard,” for example.

Next, let's create the character of the professor. Here's some code that will get us started:

```
The professor is a woman.  The description of the professor is "A
smart and kindly lady, but elderly and not too spry."  The professor
is in Coburn 301. The indefinite article of the professor is "the".
```

Here, some writing teachers will see an opportunity to teach about articles. Since “the professor” is more appropriate for our story than “a professor,” we add an assertion that controls the article. Note that, in specifying the article “the,” Inform does not follow the usual English convention for the period and its placement relative to closing quotation marks. If Inform did not make this variation, “the professor” would come out, in the story as “the. Professor.”

There's also an opportunity for work with clarity and development here, since student writers will often forget to specify a character's location. In other words, they will fail to use an assertion like

“The professor is in Coburn 301.” If a student leaves out the professor's location, the story will still compile, but the professor will not appear anywhere. She will be “off-stage,” to use Inform's terminology for such a situation. Of course, we might, under some circumstances, want a character to be off stage at first, so that we can bring her into the story later on.

Now, let's create, or “implement” the marker and the Zune.

```
The marker is a thing in Coburn 301.  The description of the marker
is "An ordinary black dry-erase marker."

The Zune is a thing.  The description of the Zune is "A slick new
music player." Understand "ipod" and "mp3 player" as the Zune.
```

Inform's assumptions about these objects, since we haven't specified otherwise, is that they are small enough to pick up and carry around, though we can easily override these assumptions when we want to. Notice that the Zune is initially off stage, since we don't want the player/character to see it until she picks up the marker for the professor. Note, also, an important consideration for clarity and development in interactive fiction: we have provided synonyms for the reader to use in referring to the Zune. Readers of IF are famously appreciative of such consideration.

Now that we've implemented some objects, let's set up our first rule for governing how the story is to be presented.

```
When play begins, say "It's another typical day in College Writing
II.  The professor is illustrating the use of thesis statements,
using a whiteboard, when she drops her marker on the floor.  She
appears to be a little hesitant to bend down and pick it up."
```

This rule will take effect only once in the story, at the very beginning. In Inform 7, “say” is an instruction to the computer to display some text on the screen. This particular rule causes only one action to happen, the “saying” of some words. We'll soon see other rules that cause a series of events to take place.

An interactive fiction does not have to include a “When play begins” rule, but nearly all IF stories do. The inclusion of such a rule helps a student writer to see the importance of a coherent beginning. We'll soon see a rule for the story's ending.

Next, let's implement a more complex rule to describe what happens when the player/character gives the marker to the professor.

```
Instead of giving the marker to the professor:
    say "The professor thanks you with notable sincerity. You have
      demonstrated the virtue of thoughtfulness.";
    move the marker to the professor;
    increase the score by 5.
```

This time, we're using an “instead” rule. Instead of what? Instead of what the model world of Inform would otherwise do. The Inform model world supports the notion of giving something to someone. It understands the verb “give” as a transitive verb that is typically followed by an object and by a

prepositional phrase starting with “to.” However, since it has no way of knowing what any given author wants to do with the action of giving, the model world, as a default, responds, to an attempt to give, with a plain-vanilla piece of text, proclaiming that the intended recipient “doesn't seem interested.”

In our case, we want to make three assertions that will take effect when the player/character gives the marker to the professor. In order to signal that we want more than one assertion to attach itself to this rule, we use a colon, rather than a comma, after “Instead of giving the marker to the professor.” Then we make our three assertions, each indented to show that all three are to follow from our “instead” rule. The first of our three assertions prints some text to the screen. This text expresses the professor's gratitude, but it doesn't do anything to move the marker. Like all assertions in a list of this sort, our “say” assertion has to end in a semicolon, odd as it may look. The second assertion handles the actual transfer of the marker to the teacher, and the third increases the player/character's score, which we use, in this story, to mark the character's demonstration of various virtues.

Now, let's implement the player/character's picking up the marker.

```
After taking the marker:
   say "As you pick up the marker, you notice that there's a Zune
     MP3 player on the floor, under your chair.  The Zune isn't
     yours, though you wish it were.  In fact, you recognize the Zune
     as the property of Matt, who's slouched beside you.";
   move the Zune to Coburn 301.
```

In its structure, this new rule looks a lot like the “instead” rule that we created earlier. This time, however, the rule does not substitute itself for Inform's default action. Instead, the rule takes effect **after** the player/character has taken the marker. Inform's default action of “taking” transfers the marker to the player's possession automatically. We don't need to create our own assertion for the transfer. The first assertion that we do make prints some text to the screen. The second assertion moves the Zune from off-stage to Coburn 301, where the player/character can now interact with it.

Some writing teachers, since they are not really trying to build better programmers, might omit the idea of the “after” rule altogether, since the same result can be achieved with an “instead” rule, like the one that follows.

```
Instead of taking the marker:
     say "As you pick up the marker, you notice that there's a Zune
       MP3 player on the floor, under your chair.  The Zune isn't
       yours, though you wish it were.  In fact, you recognize the
       Zune as the property of Matt, who's slouched beside you.";
     move the marker to the player;
     move the Zune to Coburn 301.
```

Creating the character Matt is essentially the same process as implementing the professor.

```
Matt is a man in Coburn 301.  The description of Matt is "A sincere,
if slightly lethargic young man, wearing a UMass Lowell sweatshirt."
```

And the rule for giving the Zune to Matt is similar to the rule for giving the marker to the professor.

```
Instead of giving the Zune to Matt:
     say "Matt nods. He looks a little surprised at getting his toy
      back so easily.  You have demonstrated the virtue of honesty.";
     move the Zune to Matt;
     increase the score by 5.
```

Now we need rules for a limited sort of conversation with the professor. Inform's built-in conversation system allows for asking, telling, consulting, and answering. First, we'll create a rule for the professor's posing of her question.

```
An every turn rule:
     if the score > 8:
          say "The professor turns to you and asks, 'Now, Jamie, can
          you tell us who wrote Plato's Republic?'  Unfortunately,
          you've never heard of this work."
```

Here, we've implemented an “every turn” rule. Inform will check, at the end of every turn, to see whether the conditions imposed by the rule have been met. In our case, the only condition is that the player's score be greater than eight. If the condition is met, the professor will pose her question.

Now, we need a way to implement the player's answer. We'll use an “instead” rule to handle the correct answer. Using this rule, we'll print some text to the screen, increase the score to its maximum, and end the story.

```
Instead of answering the professor that "Plato":
     say "The professor expresses appreciation of your insightful
       response.  You have demonstrated the virtue of intelligence.";
     increase the score by 5;
     end the story, saying “Congratulations! You have proven to be
virtuous.”
```

To handle incorrect answers, we'll create a similar rule that is deliberately more general than the rule for the correct answer. Inform applies a more general rule only if a more specific rule does not already apply.

```
Instead of answering the professor that something:
     say "The professor says, 'No. That's not it.'"
```

Our source code has now reached the point at which, when compiled, it produces a readable story, though not quite the story that our transcript shows. In addition, with some appropriate explanatory text, our source code would constitute an essay developed by process analysis. Still, there's one more powerful idea that we will need in order to produce the tale that our transcript tells. We need a variable. Inform allows for various sorts of variable, but let's start with a straightforward adjective.

```
A person can be pleased. A person is usually not pleased.
```

Here, we have created the adjective variable “pleased.” We've restricted this adjective to people, and

we're provided a default value for it, “not pleased.”

What can we do with this variable to shape our story? Almost anything we want. For example, if Matt is “not pleased” when the player/character has held his Zune for a certain number of turns, he might attack the player/character. On the other hand, if he is pleased, he might offer the player/character a reward for returning the MP3 player. In our transcript, though, we make somewhat subtler changes based on our variable—we change the characters' descriptions when they become pleased, by changing the source code for their descriptions as follows. Note the use of brackets to create conditions under which new text will be displayed.

```
The professor is a woman.  The description of the professor is "A
smart and kindly lady, but elderly and not too spry. [if the
professor is pleased] The professor looks pleased with you[end if]."
The professor is in Coburn 301. The indefinite article of the
professor is "the".

Matt is a man in Coburn 301.  The description of Matt is "A sincere,
if slightly lethargic young man, wearing a UMass Lowell sweatshirt.
[if Matt is pleased] Matt looks quite happy right now[end if]."
```

As our code stands now, however, neither Matt nor the professor will ever become pleased. We'll have to change our rules for giving them their objects to include a change in pleased-ness. Our assertion for making this change reads “now the professor (or Matt) is pleased.”

```
Instead of giving the marker to the professor:
    say "The professor thanks you with notable sincerity. You have
      demonstrated the virtue of thoughtfulness.";
    move the marker to the professor;
    now the professor is pleased;
    increase the score by 5.

Instead of giving the Zune to Matt:
     say "Matt nods. He looks a little surprised at getting his toy
      back so easily.  You have demonstrated the virtue of honesty.";
     move the Zune to Matt;
     now Matt is pleased;
     increase the score by 5.
```

Let's add one more tool to our programming repertoire, a numerical variable. We may decide not to use this tool in connection with the teaching of writing, but it does become useful when students try to produce more complex stories. In our example, we'll make a numerical variable to help us vary what the professor says when she asks her *Republic* question.

```
The professor has a number called questioning. The questioning of the
professor is 0.
```

Next, we create an "every turn" rule. At the end of every turn, Inform will check to see if it should carry out this rule. We'll use our numerical variable to vary what the professor says.

The first time the professor asks her question, the value of the numerical variable will be zero, its default. However, after she asks, we'll increase the value of the variable to one.

Note, also, the use of single quotation marks within the double quotation marks. For the player, Inform will display the single quotation marks as double quotes. We've added some italics here, too.

```
An every turn rule:
     if the score > 8:
          if the questioning of the professor is 0:
               say "The professor turns to you and asks, 'Now,
                  Jamie, can you tell us who wrote Plato's [italic
                  type]Republic[roman type]?'  Unfortunately, you've
                  never heard of this work.";
               increase the questioning of the professor by 1;
          if the questioning of the professor is greater than 0:
               say "The professor says, 'Jamie, I know you can
                 figure this one out. Who wrote Plato's [italic
                 type]Republic[roman type]?'"
```

We've implemented nearly all of our story, as it appears in the transcript. All that's left to add are some objects and people that aren't important to the story's interaction. These include the whiteboard, the student desks, and most of the students. We implement these elements largely to achieve a sense of completion in our interactive world. If we did not implement these objects and the player tried to examine them, the response would be a nonsensical “You can't see any such thing.” By declaring that our new items are “scenery,” we prevent the player/character from taking them. Also, we'll use the Inform concept of “understanding” to implement some synonyms to help the reader along.

```
The whiteboard is scenery in Coburn 301. The description of the
whiteboard is "An ordinary whiteboard, which the professor has been
using in her presentation."

Some desks are scenery in Coburn 301. The description of the desks is
"Ordinary desk-chair furniture." Understand "desk" and "chair" as the
desks.

Some students are scenery in Coburn 301. The description of the
students is "Ordinary collegiate scholars."
```

Finally, we'll establish two useful controls on the way the story will work. One of these controls declares the maximum score the player can achieve. The other tells Inform to punctuate series in the American style.

```
The maximum score is 15.

Use the serial comma.
```

A Process-Analysis Essay

Now the source code is complete, we can produce an essay developed through process analysis. All we'll have to do is add appropriate text to comment on the source code. The finished essay might look something like the following.

```
"A Narrative About Virtue" by Brendan Desilets

     [How can we write a computerized interactive story? Using the
authoring system called Inform 7, we can give the computer plain-
Englsh instructions about how to present almost any story. Here,
we'll develop some “source text,” which will tell the computer, step
by step, how to tell a simple interactive story. Obviously, we've
started with some text in brackets. Usually, text that appears in
brackets is not part of the instructions for the computer. Instead,
it is text intended for a human reader.]

     [Our story is about a young woman in a college class. In the
course of the story, she'll have a chance to display three
traditional virtues, thoughtfulness, honesty, and intelligence.]

     [First, we set the maximum score that the player can achieve in
this gamelike story. The score will help to document virtues that the
player/character displays.]

The maximum score is 15.

     [Next, we tell Inform to use a comma between the last two items
in a series. This use is typical in America, but not in Inform's
homeland, England.]

Use the serial comma.

     [Here we create our first noun, a room.]

Coburn 301  is a room.  The description of Coburn 301 is "A typical
college classroom, with lots of desks, a whiteboard, and a dozen or
so students."

     [Here we create an adjective variable. Once we make our
variable, any person can be either pleased or not pleased. The
default, as we declare it, is "not pleased."]

A person can be pleased. A person is usually not pleased.

     [Here we create our first character, the professor. Her
description changes, depending on whether or not she is pleased.

     Notice that we have overridden the article that Inform would
normally use in referring to the professor. We changed the article
from the default "a" to "the."  In this case, in our source code, we
```

cannot follow the usual rule of putting a period inside quotation marks. If we put the period inside the quotation marks, the story would say, "You can see the. professor."

Note, also, that, within the description of the professor, we use brackets in a way that we have not seen before. Within quoted material, such as the professor's description, brackets can be used to specify a condition under which some text will be shown to the reader.]

The professor is a woman. The description of the professor is "A smart and kindly lady, but elderly and not too spry. [if the professor is pleased] The professor looks pleased with you[end if]." The professor is in Coburn 301. The indefinite article of the professor is "the".

[Next, we create an object, in the form of a noun. The default qualities of this object allow the play to pick it up and carry it around. These defaults also cause the item to mentioned in room descriptions.]

The marker is a thing in Coburn 301. The description of the marker is "An ordinary black dry-erase marker."

[Here we create another object, but this one is an example of scenery. If an object is scenery, the player cannot pick it up, and it is not mentioned separately in the description of a room.

The reader will not be able to interact with the whiteboard, except by looking at it.]

The whiteboard is scenery in Coburn 301. The description of the whiteboard is "An ordinary whiteboard, which the professor has been using in her presentation."

[Here we create some scenery with a plural name, "desks." Also we create two synonyms, "desk" and "chair."]

Some desks are scenery in Coburn 301. The description of the desks is "Ordinary desk-chair furniture." Understand "desk" and "chair" as the desks.

Some students are scenery in Coburn 301. The description of the students is "Ordinary collegiate scholars." Understand "student" as the students.

[Rules constitute a major component of any IF story's source code. Rules control, to a great extent, how the story unfolds. This is our first rule, carried out when the story starts.]

```
When play begins, say "It's another typical day in College Writing
II.  The professor is illustrating the use of thesis statements,
using a whiteboard, when she drops her marker on the floor.  She
appears to be a little hesitant to bend down and pick it up."

     [Next, we create an "instead" rule. This rule causes something
interesting to happen when we try to give the marker to the
professor. Notice the precise use of parallel structure in this
rule.]

Instead of giving the marker to the professor:
    say "The professor thanks you with notable sincerity. You have
      demonstrated the virtue of thoughtfulness.";
    move the marker to the professor;
    now the professor is pleased;
    increase the score by 5.

     [Here we create another rule. This one takes effect after Inform
has allowed the player to pick up the marker.]

After taking the marker:
    say "As you pick up the marker, you notice that there's a Zune
      MP3 player on the floor, under your chair.  The Zune isn't
      yours, though you wish it were.  In fact, you recognize the
      Zune as the property of Matt, who's slouched beside you.";
    move the Zune to Coburn 301.

     [Here we create the Zune, in the form of a noun. Notice that the
Zune isn't anywhere when we create it. We move it to Coburn 301 when
the player takes the marker.]

The Zune is a thing.  The description of the Zune is "A slick new
music player." Understand "ipod" and "mp3 player" as the Zune.

Matt is a man in Coburn 301.  The description of Matt is "A sincere,
if slightly lethargic, young man, wearing a UMass Lowell sweatshirt.
[if Matt is pleased] Matt looks quite happy right now[end if]."

Instead of giving the Zune to Matt:
     say "Matt nods. He looks a little surprised at getting his toy
       back so easily.  You have demonstrated the virtue of
       honesty.";
     move the Zune to Matt;
     now Matt is pleased;
     increase the score by 5;

     [Next, we create a numerical variable called "questioning."
We'll use this number to vary what the professor says when she
```

questions the player/character.

The first time the professor asks her question, the value of the questioning variable will be zero, its default. However, in a rule we'll create later, we'll increase the value of the variable when she speaks.]

The professor has a number called questioning. The questioning of the professor is 0.

[Here we create an "every turn" rule. At the end of every turn, Inform will check to see if it should carry out this rule. We'll use our numerical variable to vary what the professor says.

The first time the professor asks her question, the value of the numerical variable will be zero, its default. However, after she asks, we'll increase the value of the variable to one.

Note, also, the use of single quotation marks within the double quotation marks. For the player, Inform will display the single quotation marks as double quotes.]

```
An every turn rule:
     if the score > 8:
          if the questioning of the professor is 0:
               say "The professor turns to you and asks, 'Now, Jamie,
                can you tell us who wrote Plato's [italic
                type]Republic[roman type]?'  Unfortunately,
                you've never heard of this work.";
               increase the questioning of the professor by 1;
          if the questioning of the professor is greater than 0:
               say "The professor says, 'Jamie, I know you can figure
                this one out. Who wrote Plato's
                [italic type]Republic[roman type]?'"
```

[Here we create an rule to enable the player to answer the professor's question. This rules also ends the story/game with the success of the player. Notice that Inform can understand the verb "answer." In Inform, it is also possible to introduce new verbs. We can even create verbs that can be used with particular prepositions.]

```
Instead of answering the professor that "Plato":
     say "The professor expresses appreciation of your insightful
       response.  You have demonstrated the virtue of intelligence.";
     increase the score by 5;
     end the story, saying “Congratulations! You have proven to be
virtuous.”
```

[Finally, we create a rule that applies when the

```
player/character gives any wrong answer to the professor's question.
We make this rule more general that the previous rule because Inform
will pass over a more general rule if a more specific rule applies.]

Instead of answering the professor that something:
     say "The professor says, 'No. That's not it.'"

     [Now, once we compile our story, it will be ready for our
readers.]
```

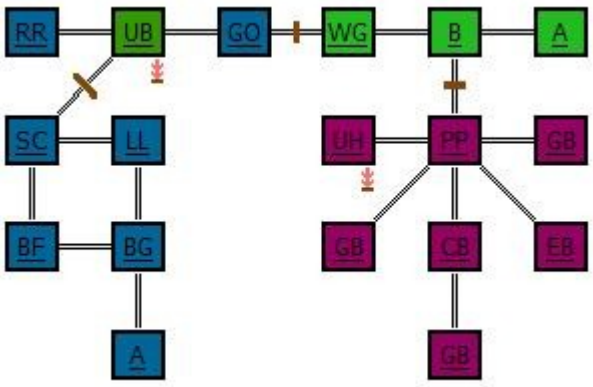
Inform 7 and Qualities of Good Writing

Inform 7 can help writers to produce good prose—prose that exhibits widely-accepted qualities of good writing, such as unity, coherence, development, and clarity.

Unity

Inform 7 helps to encourage, and even to enforce, unity in some ways that we have already seen. For example, it nudges students toward giving each story a beginning, marked by a “when play begins” rule, a middle, and at least one ending, indicated by an “end the story” assertion. In addition, Inform allows a writer to divide source code into larger and smaller pieces, called, form largest to smallest, "volumes," "books," "parts," "chapters," and "sections." The Inform compiler points directly to the names of these pieces when it complains about problems in the source text.

Unlike our sample story, many works of interactive fiction require the user to move through a great many locations. Inform helps the writer maintain the physical unity of his or her setting by generating a simple map of the story's locations. If the author has made a mistake in connecting locations, the map will usually show the error. Here, for instance, is the Inform-generated map of a fairly complex interactive fiction.



The eighteen rooms on this map connect in clear ways, with problem-solving required to get through the passages marked with a brown bar. If the writer had left a room unconnected with the rest, the map would make the error clear.

Also, Inform indexes all the people and objects in a story, showing which encloses which. Writers often check Inform's index to see how the story's parts relate to the whole. In our brief story, for example, the index looks like this:

Coburn 301 - *room where play begins*

professor - *woman*

marker

whiteboard

desks

students

Matt - *man*

yourself - *person*

Zune

This chart clearly shows an item that might violate the unity of our story, the Zune, which is not inside the story's single room. In our case, we have a good plan to add the Zune to our tightly-unified tale; but, if, in fact, we had inadvertently created an off-stage music player, the index would alert us to the problem.

Coherence

Creating an interactive fiction requires great attention to connectedness. A story that is not coherent in its use of rules and variables will often fail to compile and will never execute well. In our sample story, for example, we create the adjective variable “pleased.” If we are to use this variable, we must do so in such a way as to effectively connect with our story's rules. For example, in our rule for making the professor pleased by giving her the marker, we must specify that “now the professor is pleased,” not “happy” or “grateful.” Further, when we show the professor's pleased state in her description, we must use the bracketed text “if the professor is pleased” to connect the description with the variable, and we must complete the connection by showing the end of the text to which our condition applies, using the bracketed text “end if.”

Earlier, we noted that Inform automatically generates a story map and an outline of the story's elements. In addition to helping with unity, these devices help the author to see how coherently a story's elements connect with one another. Inform also helps the author to manage a story's locations by facilitating the grouping of the rooms into regions. In the sample map that appears above, regions are color coded.

Just as helpful is the structure known in Inform as the “scene.” Inform scenes are like the scenes of a play in that they usually have clear beginnings and endings, but these starts and stops generally depend on which actions the player/character has always taken. Suppose, for example, our “Narrative About Virtue” had a second location, called the Hallway. Suppose, further, that we want to start a new

scene when the player/character enters the Hallway, but only if she has returned the Zune to Matt, who will help her out if she has helped him. Let's say that, in this scene, the player, with Matt's help rescues a cat that is stranded on top of some scaffolding. We could create our scene with source code like this.

```
Rescue is a scene. Rescue begins when the player is in the Hallway
and the Zune is carried by Matt. Rescue ends when the player carries
the cat.
```

Some IF writers make excellent use of scenes to establish coherence, even in relatively simple stories. Inform helps writers to manage scenes well by providing a command that allows an author to see exactly when a scene starts and ends during a trial of his or her story. In addition, Inform offers an index of scenes that shows how each scene begins and ends, and indicates whether or not each scene can recur.

Development

The creation of a readable IF story requires an unusually studied sort of development, in at least five areas, characters, actions, objects, rules, and synonyms. The development of multi-dimensional characters and the creation of new actions (and the verbs that activate them) goes beyond the expertise that we would expect students to develop, if we are using Inform only as a tool to help them build their essay-writing skills. However, even in our very brief example story, we have seen the importance of developing effective objects, rules, and synonyms.

Development Through Implementing Objects

As we've noted in our creation of a sample source text, it is important for IF writers to implement the objects that they mention in their descriptions. Otherwise, their stories will generate inappropriate responses that break the immersion of their readers in the model world. Normally, the Inform compiler cannot help much with this problem; but acclaimed interactive fiction writer Aaron Reed has developed a system to customize Inform code to ensure thorough implementation of objects (Reed 77). Under Reed's system, when an IF author creates text within quotations marks, he or she put brackets around the noun that names each important object. If the author has already implemented the bracketed object, the brackets will have no noticeable effect. However, if the author has not implemented the bracketed object, the compiler will complain, thus reminding the writer that one of the story's elements needs more development.

Using Reed's “Bracket Every Notable Thing” procedure, our description of Coburn 301 would look like the following.

```
Coburn 301  is a room.  The description of Coburn 301 is "A typical
college classroom, with lots of [desks], a [whiteboard], and a dozen
or so [students]."
```

In the final version of our sample story, we have implemented the desks, whiteboard, and students. As a result, the bracketing will have no effect on the compiler's output. However, if we had forgotten to implement the desks, for instance, the computer would produce no output and would complain that it cannot understand “[desks].”

Developing Effective Rules

In the authoring of interactive fiction, developing rules sufficient presents a real challenge, especially to inexperienced writers. The Inform compiler will not generally signal problems with rule development, but even superficial reader-testing of a story usually will.

For example, to new authors, our first rule for answering the professor's question may look quite sufficient.

```
Instead of answering the professor that "Plato":
     say "The professor expresses appreciation of your insightful
      response.  You have demonstrated the virtue of intelligence.";
     increase the score by 5;
     end the story, saying “Congratulations! You have proven to be
virtuous.”
```

However, in testing the story, a reader will soon see that the rule is insufficient to deal with an incorrect answer. In evaluating a rule, especially an “instead” rule, an experienced IF writer will immediately check to see whether the rule covers all possible situations, not just the one that seems most likely. In the case of our rule, as it appears above, we have accounted for only one possibility, the correct answer. In our completed source code, we have handled the wrong answers in a separate, more general rule.

```
Instead of answering the professor that something:
     say "The professor says, 'No. That's not it.'"
```

We could also have dealt with the possibilities in a single rule, but such a rule might be more complex that we want, in modeling Inform code for our composition students.

Developing Synonyms

Interactive fiction is a difficult form of literature for most readers. One of the ways an IF author can help the reader along without damaging the reader's immersion in the story is by providing a ready supply of synonyms for important words. A lack of such synonyms often produces frustrating and unfulfilling “guess the verb” or “guess the noun” puzzles. The Inform compiler can't really help with needed synonyms, but test readers certainly can; and Inform provides a quick and easy way to create synonyms, as we did in our description of the Zune.

```
The Zune is a thing.  The description of the Zune is "A slick new
music player." Understand "ipod" and "mp3 player" as the Zune.
```

Clarity

Clarity may be the most elusive quality of good writing. The Inform compiler cannot approach the effectiveness of a careful reader in suggesting ways to clarify a text, but it can help in a number of ways, when a human reader isn't available.

Clarity in Quoted Text

In general, Inform does not actively intervene in material that appears within quotation marks. The writer usually asks only that Inform print such material on the screen, warts and all. However, in

our example story, we can see two kinds of text with quotes that the Inform compiler does check. First, consider bracket material withing quotation marks, as in our description of Matt.

```
Matt is a man in Coburn 301.  The description of Matt is "A sincere,
if slightly lethargic young man, wearing a UMass Lowell sweatshirt.
[if Matt is pleased] Matt looks quite happy right now[end if]."
```

The compiler will point out any spelling errors within the brackets, and it will report on anything that may be missing, such as a bracket or a needed phrase “[end if].” It will check to see that a variable like “pleased” has been previously defined and that it is applicable to Matt.

The compiler will also report problems with quotations with quotations, which must be marked with single quotation marks, just as they would be in standard English. If we had made the following error, our text would not compile.

```
say "The professor turns to you and asks, “Now, Jamie, can you tell
us who wrote Plato's [italic type]Republic[roman type]?”
Unfortunately, you've never heard of this work.";
```

We would need single quotation marks around the professor's words to solve this problem.

Clarity in Other Text

The Inform compiler really shines (or, some would say, becomes really picky) in checking the clarity of source code that is not within quotation marks. It will reject most spelling errors, and will require end punctuation according to rules that mimic standard English, except that Inform allows the semicolon to end an assertion. Inform checks for a comma at the end of an introductory adverbial clause, though it requires the use of a colon if such a clause is followed by more than one assertion. The Inform compiler also insists on parallel structure in lists of assertions. Such lists are very common in Inform code, as in this example from our sample story.

```
Instead of giving the Zune to Matt:
     say "Matt nods. He looks a little surprised at getting his
       toy back so easily.  You have
       demonstrated the virtue of honesty.";
     move the Zune to Matt;
     now Matt is pleased;
     increase the score by 5.
```

On the other hand, Inform does not check for capitalizing, though it almost always allows conventional use of the upper case, and it does not use the conventional comma to separate quoted text from quoted text, as in:

```
Some desks are scenery in Coburn 301. The description of the desks is
"Ordinary desk-chair furniture."
```

Inform's built-in text editor helps with clarity, too, in that it color-codes words that appear in quotation marks and in brackets.

Here's an example.

```
An every turn rule:
     if the score > 8:
          if the questioning of the professor is 0:
               say "The professor turns to you and asks, 'Now,
                 Jamie, can you tell us who wrote Plato's
                 [italic type]Republic[roman type]?'  Unfortunately,
                 you've never heard of this work.";
               increase the questioning of the professor by 1;
          if the questioning of the professor is greater than 0:
               say "The professor says, 'Jamie, I know you can figure
this one out. Who wrote Plato's [italic type]Republic[roman type]?'"
```

“Commented” text, which the compiler ignores, appears in green, as in

```
[This is our first rule. It is carried out when the story starts.]
```

Inform and the Writing Process

In addition to providing students with some insights into the quality of the writing, Inform can help them to understand the phases of the writing process, though some of these phases take on some new wrinkles in the creation of interactive fiction.

Prewriting

Even the simplest of IF stories, such as the one we've used as an example here, require some explicit planning. This planning has to address the usual concerns of narrative writing, such as plot, setting, and character, and some concerns that address the peculiarities of interactive fiction.

In planning a first work of interactive fiction, it makes sense to look for brevity. Even our super-brief example story requires 575 words of source text, not counting comments, and getting source text to work can be a real challenge for any new author. If we add enough commenting to make the story comprehensible to an intelligent, but previously uninformed, reader, we approximately double the length. In addition, a new IF writer should think about what does, and does not, work well in an all-text interactive medium. IF stories do well with exploration, discovery, and problem-solving. They are not well suited to quick-twitch reactions.

After finding a suitable story topic, a new author generally does well to outline, or map, the basics of the story's plot, characters, objects, puzzles, and setting. The rigors of writing a story that requires computer programming work strongly against the composing styles of students who resist planning. An unplanned story is virtually impossible to implement. Often IF planning goes beyond an outline. If the story has a half dozen or more rooms, the writer will probably produce a map as part of his or her prewriting. In a longer story, the author may create a list of scenes.

Drafting

After prewriting, an IF writer often produces a detailed prose version of the story, or at least of

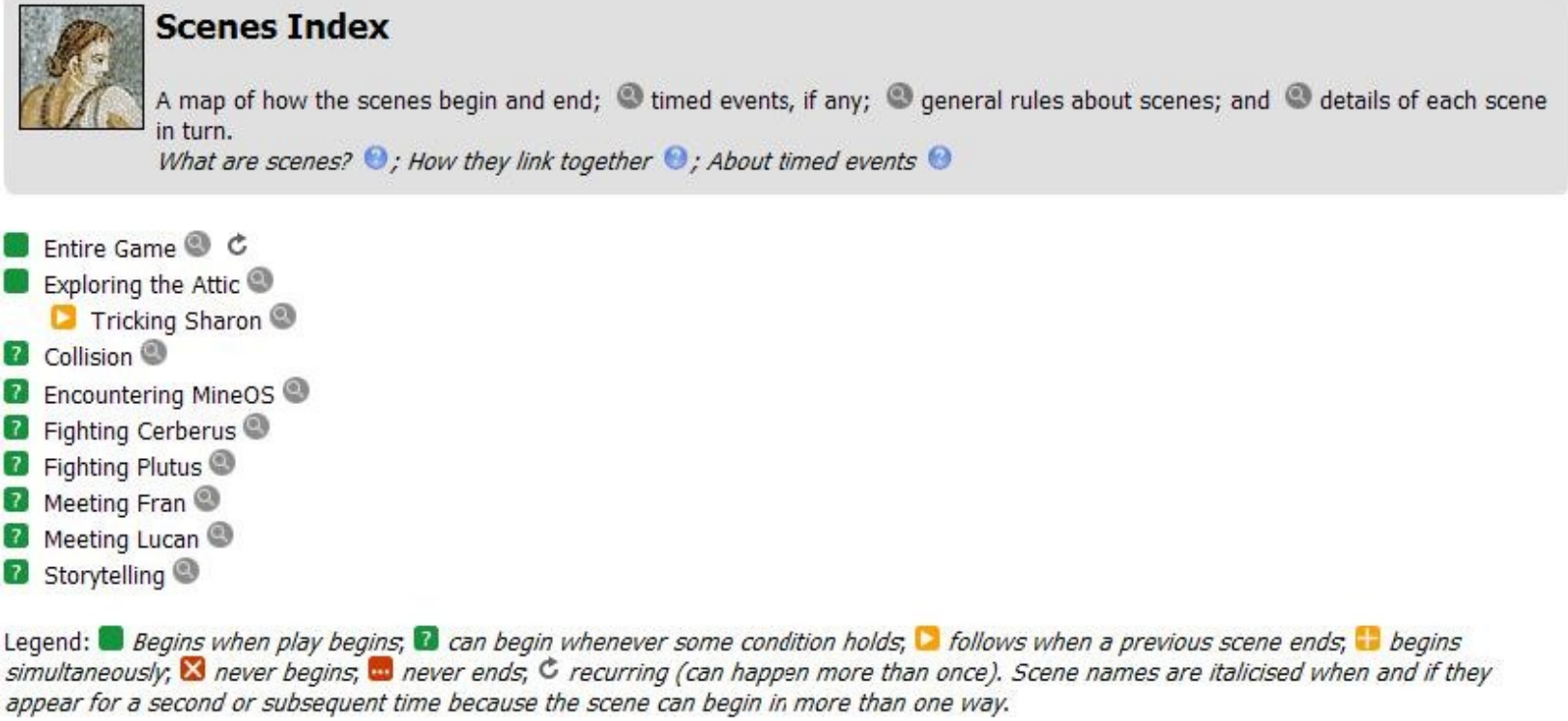
one possible run-through of the piece. This non-interactive draft usually takes the form of an imagined transcript of a session with the story. In the case of a longer story, the author may draft a scene and implement that scene in Inform, before going on the next scene (Rees).

Of course, the first draft of an interactive fiction is not complete until it has taken an interactive form. For this reason, drafting in IF requires creating source text and testing the source text by trying to compile it. When the compiling fails, the writer will typically jump ahead to the revising and editing stages of the writing process until the text compiles, before getting back to drafting. This sort of alternating among drafting, revising, and editing may run against the grain for many teachers of the writing process, while others will find this sort of shifting quite natural. In any case, no one can deny that producing an interactive fiction requires active revising and editing. Without revising and editing, the source code will simply never compile.

Revising

Even after the source code compiles, the revising stage of writing an IF story has a long way to go. “Alpha” testing, by the writer herself invariably reveals many problems that need correcting, and “beta” testing, by others, will reveal even more issues, ranging from difficulties in character development to rules that don't work as intended. Inform offers some tools that can help.

As noted earlier, Inform generates a map of the story's model world to help with revisions, and it also creates an outline of all the story's rooms, characters, and objects. To help with scenes, Inform indexes all of a story's scenes and their properties, as in the following example.



Inform also provides a testing command, “scenes,” which causes the playing of a story to note each

scene change when it occurs.

Rules often require revision, partly because they are difficult to craft in themselves, and partly because they often interact with one another in complicated ways. The “rules” testing command is, therefore, very useful. This command causes the story's output to list, every turn, each rule that applies. The results of the “rules” command can be a bit confusing at first, because it lists not only the applicable rules that the author has created but also the underlying, “standard” rules of the Inform system, which which the writer's rules may conflict. Here's an example from our sample story.

```
>rules
Rules tracing now switched on. Type "rules off" to switch it off
again, or "rules all" to include even rules which do not apply.

>take marker
[Rule "can't take yourself rule" applies.]
[Rule "can't take other people rule" applies.]
[Rule "can't take component parts rule" applies.]
[Rule "can't take people's possessions rule" applies.]
[Rule "can't take items out of play rule" applies.]
[Rule "can't take what you're inside rule" applies.]
[Rule "can't take what's already taken rule" applies.]
[Rule "can't take scenery rule" applies.]
[Rule "can only take things rule" applies.]
[Rule "can't take what's fixed in place rule" applies.]
[Rule "use player's holdall to avoid exceeding carrying capacity
rule" applies.]
[Rule "can't exceed carrying capacity rule" applies.]
[Rule "standard taking rule" applies.]
[Rule "After taking the marker" applies.]
As you pick up the marker, you notice that there's a Zune MP3 player
on the floor, under your chair.  The Zune isn't yours, though you
wish it were.  In fact, you recognize the Zune as the property of
Matt, who's slouched beside you.
```

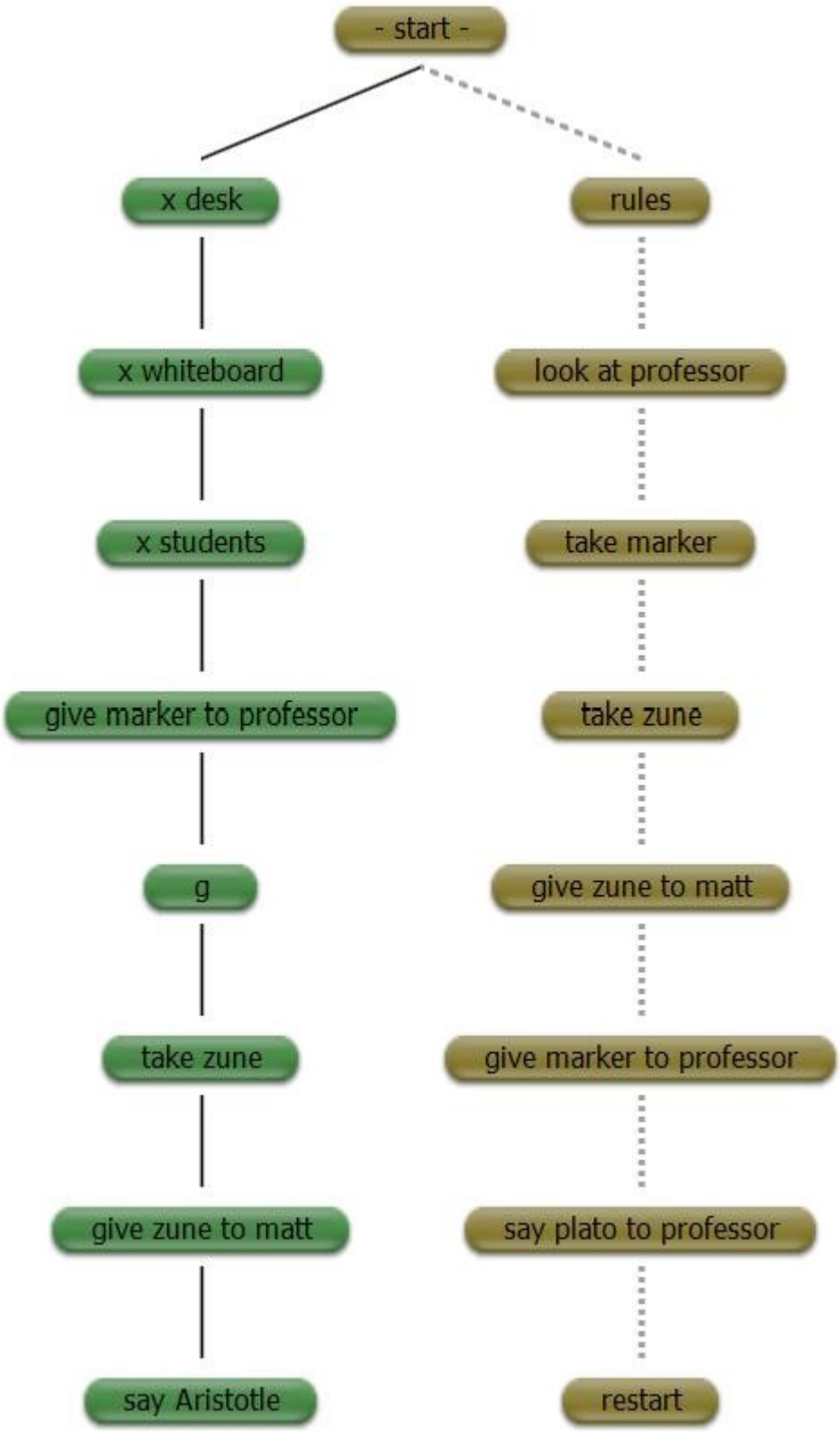
Variables often require revising, too, since it's easy to make mistakes in applying them to people and objects. Frequently, an author may think that a variable adjective, such as our “pleased” should be applying when the execution of the story indicates otherwise. To help with this sort of revision, Inform provides the testing command “showme,” which reveals all of an object's properties. Consider the following transcript, which shows a change in whether the professor is pleased and in what she carries.

```
>showme professor
professor - woman
location: in Coburn 301
unlit; inedible; portable
female
printed plural name: women
carrying capacity: 100
printed name: professor
indefinite article: the
```

```
description: A smart and kindly lady, but elderly and not too spry.
questioning: 0

>give marker to professor
The professor thanks you with notable sincerity. You have
demonstrated the virtue of thoughtfulness.

[Your score has just gone up by five points.]

>showme professor
professor - woman
    marker
location: in Coburn 301
unlit; inedible; portable
female; pleased
printed plural name: women
carrying capacity: 100
printed name: professor
indefinite article: the
description: A smart and kindly lady, but elderly and not too spry.
The professor looks pleased with you.
questioning: 0
```

In a typical writing class, would students use testing tools commands like “scenes,” “rules,” and “showme”? Probably not, until (or unless) they needed them. However, students would, almost to a person, use the most useful revising tool of all, Inform's “skein.” In revising an interactive fiction, writers often need to replay the story to a particular point. This sort of replay can be repetitive and exhausting, but not if one uses the skein. Surprisingly, the skein keeps track of all the moves of every playing through of a particular story. These attempts at working through the story can become so numerous that Inform allows for trimming the skein from time to time.

Here's a look at a trimmed version of the skein for our example story.



When a writer is viewing the skein in Inform, he or she can double click on any node (or “knot”), causing the story to play automatically to that point. If the author has been trying to remedy a problem

that occurs at the chosen knot, the skein's replay will usually reveal whether the remedy has succeeded. If the remedy hasn't worked, the author can use Inform's other revision tools, or he or she can study the game's output to see where the problem first appears. Or the author can try Inform's “transcript” tab, which coordinates closely with the skein to help the author compare various replays, some of which will likely seem more correct than others.

And, best of all, an IF author can depend on the very supportive interactive fiction community for help. Some genuinely expert writers offer free and responsive advice at the Interactive Fiction Forum (http://www.intfiction.org/forum/).

Editing

We've already seen that Inform's color-coded text editor helps with paired punctuation marks, such as brackets, and that the Inform compiler flags a variety of issues that call for editing. These include spelling errors, and problems with various punctuation marks, including quotation marks, commas, and end marks. Most Inform authors, especially inexperienced ones, will feel that they have to resolve the compiler's complaints as they create source text, rather than postponing their editing until later.

Still, most of the editing that goes into an interactive story is much like the editing of any other piece of text. It requires close attention, benefits enormously from a reader's help, and occurs, most profitably, after drafting and revising.

Publishing

Inform offers exciting options for publishing, both in the sense of producing a polished final draft and in the sense of reaching readers. Inform allows a writer to produce a story that does not include testing commands, such as “rules” and “scenes.” Such commands only get in a reader's way. Quite easily, an author can include, in her final, compiled draft, cover art for her story, a booklet that introduces interactive fiction, a website about her story, a link that allows a reader to experience the story over the Web, and a walkthrough.

The interactive fiction community offers an active group of readers, too, reachable through the Interactive Fiction Forum (http://www.intfiction.org/forum/). The Interactive Fiction Archive (http://www.ifarchive.org/) houses thousands of IF stories an related material. New contributions to the archive often attract readers and, sometimes, reviewers. Competitions for interactive fiction occur often and help to provide readers for new stories. The most prominent of these is the annual IF Comp (http://www.ifcomp.org/).

Advantages of Inform 7

It seems, then, that, from a writing teacher's viewpoint, Inform 7 has some real advantages of other programming languages. It is easy to learn. It works with natural-language source text that takes the form of an essay developed by process analysis. It produces interesting prose output, in the form of interactive fiction. Its compiler helps to flag a variety of problems, ranging from spelling errors to completeness of rules. And its built-in text editor, with its color-coding, helps with revising and proofreading. These advantages feed into ideas and processes that have strong support among writing teachers and researchers—ideas like the qualities of good writing and processes like prewriting, drafting, and revising.

But real teachers of writing may have reservations about the time and effort needed to get students started with Inform 7. What would constitute a truly minimalist approach that would allow students to produce a brief process analysis draft very quickly, perhaps in one class period?

One minimal sort of interactive story is the simple riddle. A transcript of such a story might look like the following.

```
A Riddle

An Interactive Fiction by Brendan Desilets

Release 1 / Serial number 101120 / Inform 7 build 6E72 (I6/v6.31 lib
6/12N)


Riddle Room

This is a room with a big sign and a number of toys scattered around.


The big sign poses a riddle. "What is tall as a house, round as a
cup, and all the king's horses can't draw it up? Pick up the correct
toy to answer the riddle?"


You can also see a the rubber duck, a Tonka truck, a Barbie doll, a
miniature robot, an old cell phone, a toy well, a wooden plane, a
plastic hammer and a picture book here.


>take duck



    *** Sorry. That's not it. ***




Would you like to RESTART, RESTORE a saved game, QUIT or UNDO the
last command?

> restart


A Riddle

An Interactive Fiction by Brendan Desilets

Release 1 / Serial number 101120 / Inform 7 build 6E72 (I6/v6.31 lib
6/12N)
```

```
Riddle Room
This is a room with a big sign and a number of toys scattered around.

The big sign poses a riddle. "What is tall as a house, round as a
cup, and all the king's horses can't draw it up? Pick up the correct
toy to answer the riddle?"

You can also see a the rubber duck, a Tonka truck, a Barbie doll, a
miniature robot, an old cell phone, a toy well, a wooden plane, a
plastic hammer and a picture book here.

>take well
Well done! You've solved the riddle.

    *** You have won ***
```

This minimal story requires very few Inform 7 skills. The writer has to create a room, implement some objects, and develop an instead rule or two. The source text would look like this:

```
"A Riddle" by Brendan Desilets

Use no scoring.

The Riddle Room is a room. "This is a room with a big sign and a
number of toys scattered around."

The sign is in the Riddle Room. "The big sign poses a riddle. 'What
is tall as a house, round as a cup, and all the king's horses can't
draw it up? Pick up the correct toy to answer the riddle?'"

The rubber duck, the Tonka truck, the Barbie doll, the miniature
robot, the old cell phone, the toy well, the wooden plane, the
plastic hammer, and the picture book are in the Riddle Room.

Instead of taking the toy well:
     say "Well done! You've solved the riddle.";
     end the story, saying “Congratulations! You have won.”
```

```
Instead of taking something:
     end the story saying "No. That's not it."
```

After a brief study of this riddle story and its 134-word source code, students should be able to create their own riddle stories. Then, their challenge will be to add enough clear comments to make the source code intelligible. The whole essay would end up at around 300 words, quite manageable for a one-class exercise.

Table 1--Inform 7 Versus Four Other Programming Languages

Language	Easy to Learn	Natural Language Source Code	Natural Language Output	Readable Process Analysis Essay
Logo	Yes	No	Not Usually	No
Inform 6, TADS	No	No	Yes	No
Adrift	Yes	No	Yes	No
Inform 7	Yes	Yes	Yes	Yes

Inform 6 and TADS (The Adventure Development System) are programming languages that help authors to produce interactive fiction. Adrift is a menu-based system for producing works of interactive fiction.

Works Cited

Desilets, Brendan. "Logo and Extended Definition." *Journal of Teaching Writing* 5.1 (1986): 43-50. Print.

Montfort, Nick. *Twisty Little Passages: An Approach to Interactive Fiction*. Cambridge, Massachusetts: MIT Press, 2003. Print.

Nelson, Graham. *"Inform 7: Interactive Fiction from Natural Language."* 2006. rec.arts.int-fiction. Web. November 22, 2010.

Papert, Seymour. *Mindstorms: Children, Computers, and Powerful Ideas*. New York, New York: Basic Books, 1980. Print.

Reed, Aaron A. *Creating Interactive Fiction with Inform 7*. Boston: Cengage Learning, 2011. Print.

Rees, Gareth. "Game Design and Game Analysis." *Inform Programming.* October 1995 1995.Web. November 22, 2010 <http://www.doggysoft.co.uk/inform/write/desgn.html>.

Get Lamp. Dir. Scott, Jason. Prod. Jason Scott. Jason Scott Productions, 2010. DVD.