Interactive Story Writing in the Classroom: Using Computer Games

Mike Carbonaro
Department of Education, University of Alberta
Edmonton, Alberta, Canada T6G 2G5
Mike.Carbonaro@ualberta.ca

Maria Cutumisu, Matthew McNaughton, Curtis Onuczko, Thomas Roy, Jonathan Schaeffer, Duane Szafron
Department of Computing Science, University of Alberta
Edmonton, Alberta, Canada T6G 2E8
{meric,mcnaught,onuczko,troy,jonathan,duane}@cs.ualberta.ca

Stephanie Gillis
Holy Trinity High School
Edmonton, Alberta, Canada T6K 4A5
sgillis@cs.ualberta.ca

Sabrina Kratchmer
Ardrossan Junior Senior High School
Ardrossan, Alberta, Canada T8E 2M8

ABSTRACT

Interactive story writing is a new medium for creative expression. The story “writer” uses a computer game (such as BioWare’s Neverwinter Nights) to create an interactive story where the “reader” is an active participant. The state of the art is that the story (plot, character behaviors, character interactions, conversations, etc.) is specified by writing scripts. Unfortunately, scripting is too low level for non-programmers. ScriptEase is a tool for writing interactive stories in role-playing games that frees the author from doing explicit computer programming. Stories are created by selecting and customizing familiar patterns. From this specification, ScriptEase automatically generates Neverwinter Nights scripting code. To test the usability of ScriptEase, the tool has been used as an aid to help with the short story unit of a Grade 10 Alberta high school English curriculum. This paper describes ScriptEase and reports on our experience in using it in the classroom.

Keywords

Interactive story writing, scripting, role-playing games, Neverwinter Nights.

INTRODUCTION

Computer games offer a new medium for creative writing – immersive stories where the “reader” is an active participant in the story. These stories are rich in visual and audio texture. Decisions made by the reader influence how the story unfolds and possibly even changes the outcome. Traditional pen-and-paper story writing has the author specify everything textually. In interactive stories, the "writer" uses computer tools to create visual representations of an imagined world. Vibrant colors and visual objects replace textual adjectives and vivid descriptions.

The last five years have seen interactive story-writing technology mature to the point where it has become widely popular. Unfortunately this technology requires the writer to also write sophisticated computer programs to control the interactions between the game components. For example, when the "player" steps on a particular stone in a corridor, scripting code must be written to recognize that the stone has been stepped on and to trigger an appropriate trap. Writing scripts to control this level of detail in a story is very burdensome to the writer, and only able to be done after extensive training. The goal of our research is to create an environment that enables non-programmers to write interactive stories without the need to provide scripts to control this level of detail.

Although game play can be used as an educational experience, we are more interested in game design than game play [3]. This is a relatively new endeavor [7]. The focus of the research described in this paper is to use interactive story writing as a new vehicle for creative expression. In a traditional story, the world is created with words, using descriptive prose, and the story is told with words, through narrative prose. In an interactive story, the world is "painted" with a computer-aided design tool and the story is told dynamically as the player character (PC) navigates through the world. There are potential benefits to using interactive story writing in the classroom. First, students can improve the skills necessary to effectively use an increasingly important communications medium. Second, they will learn important logical thinking skills, similar to computer programming, but in an environment that does not have the stigma of computer programming. Finally, this new communications medium provides an alternative mechanism for creative expression that may allow students to improve their expressive skills.

To reduce the scale of the problem to a manageable level, we decided to focus on computer role playing games (CRPGs). In our project, interactive stories are "written" by creating game adventures for BioWare's Neverwinter Nights (NWN) computer game system [1]. Neverwinter Nights was released in 2002 to critical acclaim, winning multiple awards (86 at last count). It was novel in that it provided a complete toolset for writing interactive stories in the NWN framework. The accompanying Aurora toolset has the capability to create story backdrops and scenery, and to populate the scenes with characters and supporting props. The scripting language NWScript can be used by the writer to specify plot components, character/prop behaviors, and their interactions. Scripting languages attempt to lessen the programming burden by presenting the user with a simplified specification language – but it is still too close to computer programming. Programming (writing) interactive games with such tools is slow, cumbersome, and fraught with error. Currently, no other game offers a better or more complete package for writing interactive stories.

Neverwinter Nights is a community game with over two million registered users at the BioWare site. Thousands of people create NWN stories and post them on the web for others to play. For example, the Neverwinter Nights Vault site hosts more than 3,000 adventures [6]. The most popular community adventure has been downloaded more than 250,000 times and the tenth most popular one has been loaded almost 100,000 times (as of March, 2005).

If the interactive story-writing community is to grow, the tools used to create these stories must be improved. They must cater to non-programmers. This paper discusses ScriptEase, a high-level tool for writing interactive stories that frees the author from doing explicit computer programming [8]. In ScriptEase, the user specifies the story components by selecting from a palette of familiar story patterns (e.g., stepping on a trap and having something happen), and then customizing them for their story (e.g., specifying which trap, and what is to happen when the trap is sprung). The goal of the ScriptEase project is to create a simple tool that enables a game designer (programmer or non-programmer) to generate a complex computer role-playing story with minimal effort.

To validate claims that ScriptEase is easy to use for non-programmers, we describe the first time it has been used in the classroom (a Grade 10 English class). In this pilot, the students learned to use the Neverwinter Nights and ScriptEase toolsets to write interactive stories. These stories were graded and were included as part of the assessment for the English course.

This paper starts off with a description of the current state-of-the-art in interactive story writing: manual scripting. Non-programmers often find manual scripting very difficult and get frustrated during the story design and develop process. The paper then discusses a new approach to interactive story writing developed by our research team called ScriptEase. We show how ScriptEase's pattern-based approach can be used to quickly construct intricate stories without writing any scripting code. Following that, we discuss the differences between interactive story writing and traditional pen-and-paper story writing. The emphasis is on the pedagogical differences that might be seen in a classroom setting. We then describe our pilot classroom study and provide insights on what was learned.

MANUAL SCRIPTING

NWScript is the language used for scripting stories in Neverwinter Nights. The language strongly resembles the C programming language. It is event-based, which means that when an event happens to an object (e.g., a treasure chest would have an event occur when it is opened or closed or when an item is added or removed from it), the script associated with that event is executed. For example, consider the common occurrence of taking an item out of a chest or putting an item into a chest. The object under consideration is the treasure chest, and the event that happens to it is that a shield is put into it. Figure 1 illustrates the NWScript code that a user might write to handle this event. When a shield is placed in a chest, the shield is destroyed, a nearby door is opened, and the character gains so-called experience points. While this might be easily readable by programmers, it is very confusing for non-programmers.

Essentially, the entire story has to be written using scripts similar to the one described above. This includes:

- interesting encounters with objects (e.g., having something exciting happen when an item in a chest is removed),
- non-player character (NPC) behaviors (e.g., having a character guarding a chest behave like one would expect a guard to behave),
- conversations (the text said by individuals has to be specified, and any actions that happen when text is spoken must be specified—for example when a PC says "No I won't give you my gold", the NPC may be required by the story to attack the PC), and
- the plot (you can only enter the room once you have discovered the secret password).

Neverwinter Nights provides a set of commonly occurring canned scripts that the user can adapt during the story-writing process. Although these canned scripts can be helpful in speeding up the story-writing process, they can also have the negative affect of limiting the story-writer's creative ability. Essentially, for most interactive story writers, this will constrain their capacity to express themselves to only those events that are easily accessible in the canned scripts. Overcoming such constraints is an important design consideration when developing a new scripting environment for story writers.

```
void main()
{
   object oItem = GetInventoryDisturbItem();
   int nItemBase = GetBaseItemType(oItem);
   if(GetLocalInt(OBJECT_SELF,"NW_L_M1S1Opened") == FALSE
            && GetTag(oItem) == "M1S1Shield" )
   {
      DestroyObject(oItem);
      object oDoor =
            GetNearestObjectByTag("M1Q5F03_M1Q5J1");
      AssignCommand(oDoor,ActionOpenDoor(oDoor));
      SetLocked(oDoor,FALSE);
      SetLocalInt(OBJECT_SELF,"NW_L_M1S1Opened",TRUE);
      RewardXP("m1q1_Never",50,GetPCSpeaker());
   }
}
```

Figure 1: Disturbing an item in a chest using NWScript.

Computer role-playing games involve large virtual worlds with thousands of characters and objects, each of which has to be scripted to obtain the desired story. Each character or object may need multiple scripts (one for each event that it responds to). Writing the scripts can be a tedious and error-prone process. Testing scripts is difficult because of the non-linear nature of interactive stories. In fact, the only really effective way to test scripts is to play through the portion of the game relevant to that script. Unfortunately, this is labor intensive and, hence, expensive.

Most Neverwinter Nights story writers are not programmers. There are few choices available to the writers for creating their story. They either use NWScript (and, hence, learn to program) or use a tool that reduces the programming burden. The Lilac Soul tool is widely used, but it provides only a mechanism for writing script fragments that then must be manually pasted into the appropriate event handlers [4]. In addition, although high-level menu commands such as "*Give items(s)/XP/gold*" generate scripting code such as "`RewardPartyGP(21, oPC, FALSE)`", there is no way to easily remove generated code from a script without trying to figure out what script code is generated by what menu commands. In practice, users choose one of these two options, and then turn to the community for help by posting to one of the NWN scripting forums. There have been almost 150,000 scripting-related postings to the BioWare forums (as of March 2005).

For non-programmers, scripting remains a mystery. Program code, like NWScript, is confusing and non-intuitive. Users tend to think in familiar high-level terms (like open a chest), not at the low-level programming details (such as calling `GetNearestObjectByTag()`). Figure 2 shows a sample request for help posted to a scripting forum (the name has been removed). The request is simply expressed in non-programming terms and the (possible) solution comes back in programming terms—a communications disconnect. The response to the request is helpful, to a point. As a non-programmer, one has to trust the validity of the code given. However comments of the form "`something like this`" leave the non-programmer confused, especially when the code as given does not work. In this example, the documentation is in error—it should be "`OnOpen`", not "`OnOpened`".

Query

```
Author: #####
Subject: Skeleton Spawn when chest opened

Is there a way to make some skeletons spawn when
you open a treasure chest?
```

Response

```
Author: #####
Subject: Re: Skeleton Spawn when chest opened

Put something like this into the OnOpened Script
of the chest.
void main() {
  CreateObject(OBJECT_TYPE_CREATURE,
               "MY_RESREF",
               GetLocation(OBJECT_SELF));
}
where MY_RESREF is the BluePrintResRef of the
creature you want to spawn.
```

Figure 2: Sample user query (1).

Figure 3 further illustrates the communications disconnect. Again, comments like "`I think`" (implying uncertainty) and "`rip the scripts`" (imprecise specification) cast doubt on the veracity of the proposed solution. The second response in Figure 3 is relatively rare—a precise specification of the changes needed. Unfortunately, almost all such postings come with the caveat "`Code presented in this past has not been compiled or tested!`".

The problems do not end there. The non-programmer needs to know where to put the script (or fragment of script). Code fragments often must be composed or assembled and the scripter often does not understand how the parts go together. Invariably, these scripts must be customized. A fragment has placeholder values requiring customization (replacing the object(s) in the script with the object(s) needed by the user). These, and many other problems, make it challenging for a non-programmer to script an interactive game story.

SCRIPTEASE

By computer game standards, NWScript is a state-of-the-art scripting language. However, the scripting language is difficult for non-programmers to learn (see Figure 1). It closely resembles the C programming language, requiring the user to understand concepts such as functions, types, and variables, as well as a large library of necessary routines. This is a serious impediment to making the story creation capabilities accessible to a non-technical audience.

ScriptEase is a scripting tool developed at the University of Alberta [8, 5] that generates NWScript for Neverwinter Nights. The program provides a menu-driven, natural language interface that is used to specify the story. From the user specifications, the tool automatically generates the appropriate NWScript code to perform the desired actions.

Query

```
Author: #####
Subject: Help me newb

I need a script for pulling a lever to make a door open.
```

Response 1

```
Author: #####
Subject: Re: Help me newb

Use the Lever to assign ActionOpenDoor (I think) to the
door - or signal a user defined event to the door, and
then have the door open itself - Alternatively load up
the prison floors from Chapter 1 of the official campaign
and rip the scripts from there.
```

Response 2

```
Author: #####
Subject: Re: Help me newb

1. Setup the door to have the tag "my_door" (or whatever)
2. Place the lever near the door
3. Edit the levers OnUsed script to read

   object oDoor = GetNearestObjectByTag("my_door");
   int nLocked = GetLocked(oDoor);
   // if the door is locked, unlock it, and vice versa
   SetLocked(oDoor, !nLocked);
   if (nLocked) { // if it was locked it is now unlocked
      AssignCommand(oDoor,ActionOpenDoor(oDoor));
   } else { // else close and lock
      AssignCommand(oDoor,ActionCloseDoor(oDoor));
      AssignCommand(oDoor,
      ActionDoCommand(SetLocked(oDoor,TRUE)));
   }
```

Figure 3: Sample user query (2).

Writing stories in ScriptEase is accomplished using patterns. The user specifies a pattern and then customizes it to suit their needs. For example, a frequently occurring pattern in fantasy games is to open a chest and have something happen. The user selects this pattern and then is presented with a window identifying the parameters to be set. The parameters identify the participants in the pattern (e.g., which chest does the pattern apply to), the pattern-related actions that are appropriate to the plot (a magical spell that is cast on the PC, a statue that animates, teleporting the player's character to another location, etc.), and special effects (e.g., a visual effect that occurs when the chest is opened).

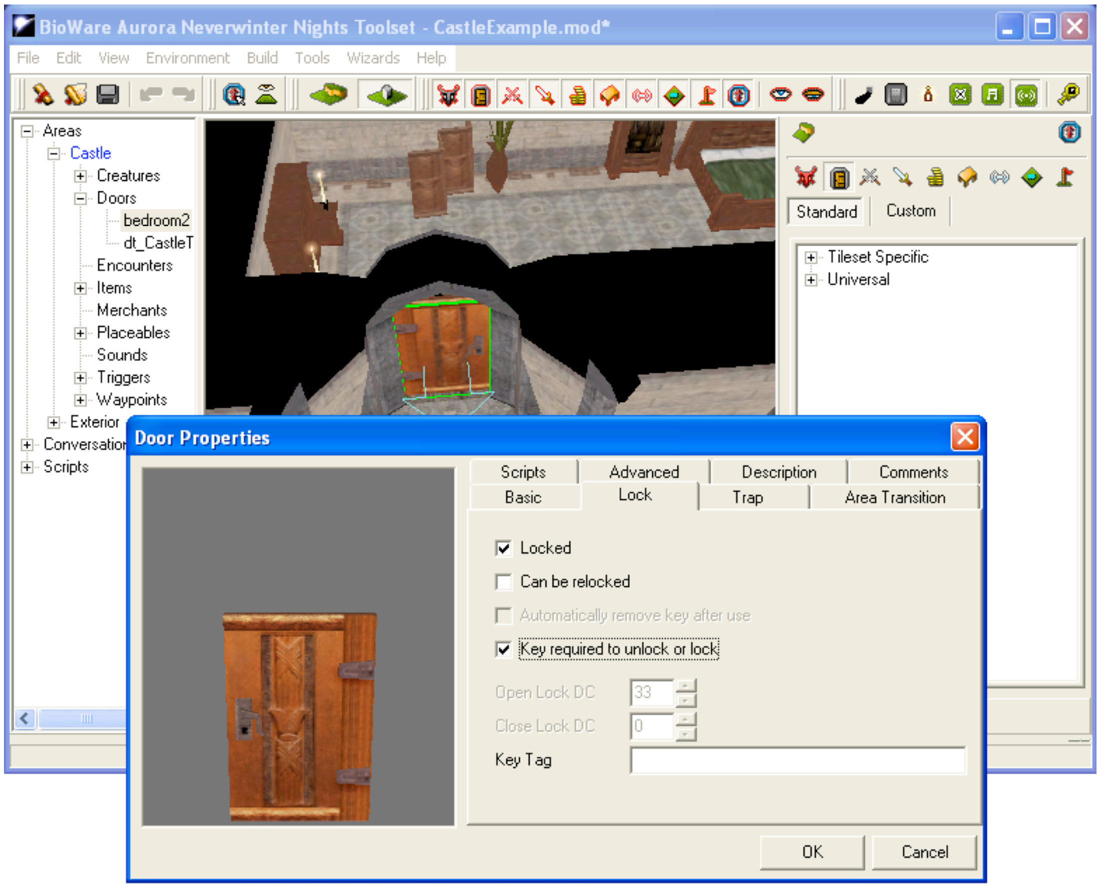
A story is written by adapting existing patterns to specify the plot, character and prop interactions, character behaviors, and conversations. We now present an example to show the step-by-step process of writing a scene from an interactive story using ScriptEase.

The *Container Disturb (specific item) toggle door* pattern is a popular pattern for controlling the plot of a story. Some writers used it to deny access to a room by locking the door and not providing a key to that door anywhere in the story. Instead, they may place a chest in another room and put a specific item (such as a particular book) into that chest. The writer can then apply the pattern so that removing the book from the chest automatically causes the locked door to unlock and open. To insert this pattern into a story, a writer begins by using the Aurora toolset to lock the door of interest. Figure 4 shows the story (in a file labeled *CastleExample.mod*) opened in the Aurora toolset. To lock the door (labeled *bedroom2*) for a specific key, the writer selects the door, opens the *Door Properties* dialog box and selects both the *Locked* and *Key required to unlock or lock* checkboxes.

The Aurora toolset is then used to create a chest (labeled *BookChest*) and place it into a nearby room. The writer then opens the *Inventory* of the chest and drags a particular book (labeled *The Origin of Magic*) into the chest. Figure 5 shows the chest, its properties dialog box, its inventory and the book that has been dragged into the inventory.

So far, the story writer has used the Aurora toolset to populate the story with appropriate props to tell this part of the story. This is equivalent to using descriptive prose in a traditional story to describe the scene and set the stage for the action. At this point, the story writer will use a ScriptEase pattern instead of writing narrative prose that describes how the story's protagonist removes the magic book from the chest to unlock and open the door. The story writer opens the story (labeled *CastleExample.mod*) in ScriptEase and creates a *New Specific Encounter* called the *Container Disturb (specific item) toggle door* pattern as shown in Figure 6.

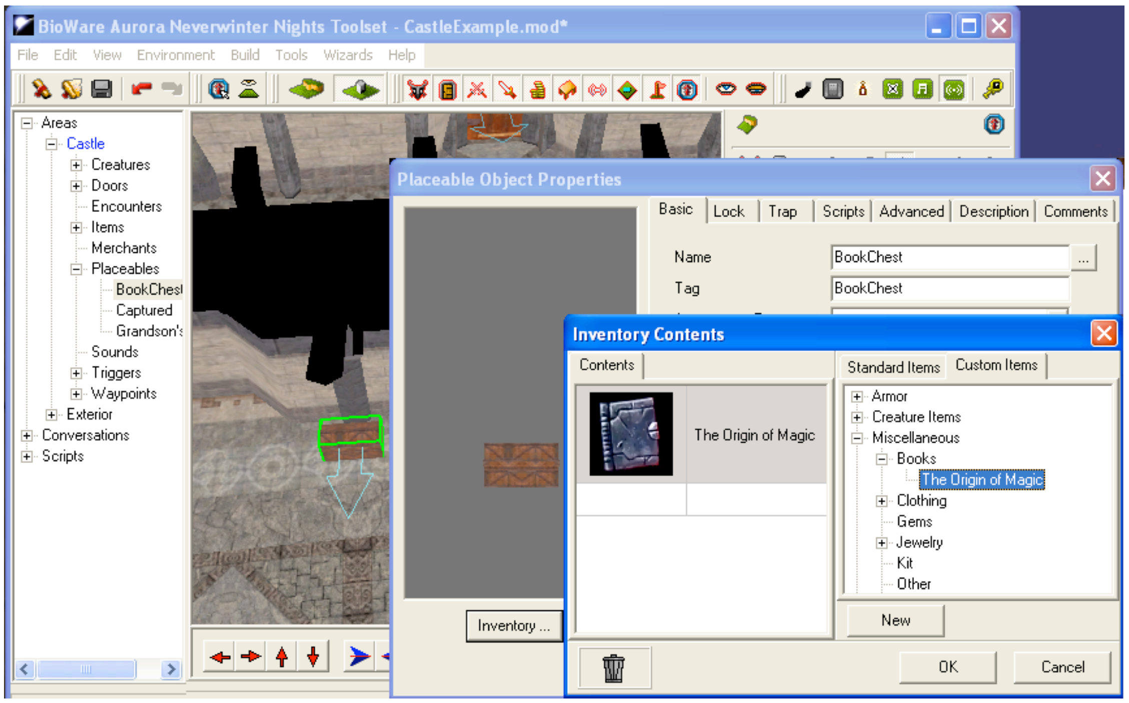
Each pattern has some roles associated with it that must be played by particular objects in the scene of the story being written. For this story, the writer selects each of the three roles associated with this pattern: *The Container*, *The Specific Item* and *The Door* and selects the objects that will play these roles: *BookChest*, *The Origin of Magic* and *bedroom2*, respectively. For example, Figure 7 shows how the user selects the tab for *The Container* role, clicks on the *Pick…* button and then selects the *BookChest* as the container of interest from a dialog box that is very similar to the (by now) familiar Aurora toolset dialogs. Note that we have overlaid a type system on the NWN objects to reduce errors. For example, only the creature and placeable icons at the top of the *Pick a blueprint* dialog are not grayed out, since a "container" is an object that has an inventory and therefore must either be a creature or a placeable, as opposed to a door, or an item, etc. that have no inventories. Our type system (even though the writers do not know it is a type system) helps to reduce errors by only allowing the appropriate type of object to be picked for any particular pattern role.



☑ Locked
☐ Can be relocked
☐ Automatically remove key after use
☑ Key required to unlock or lock

Open Lock DC 33
Close Lock DC 0
Key Tag

Figure 4: Example story opened in the Aurora toolset.



Name BookChest
Tag BookChest

Contents: The Origin of Magic

Figure 5: The chest and its properties in the Aurora toolset.

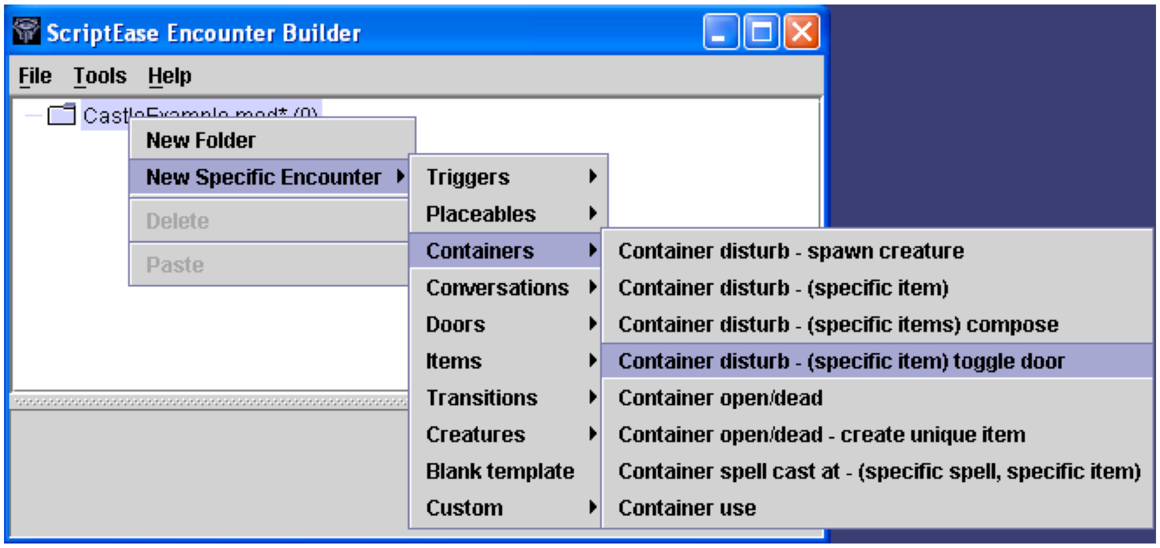
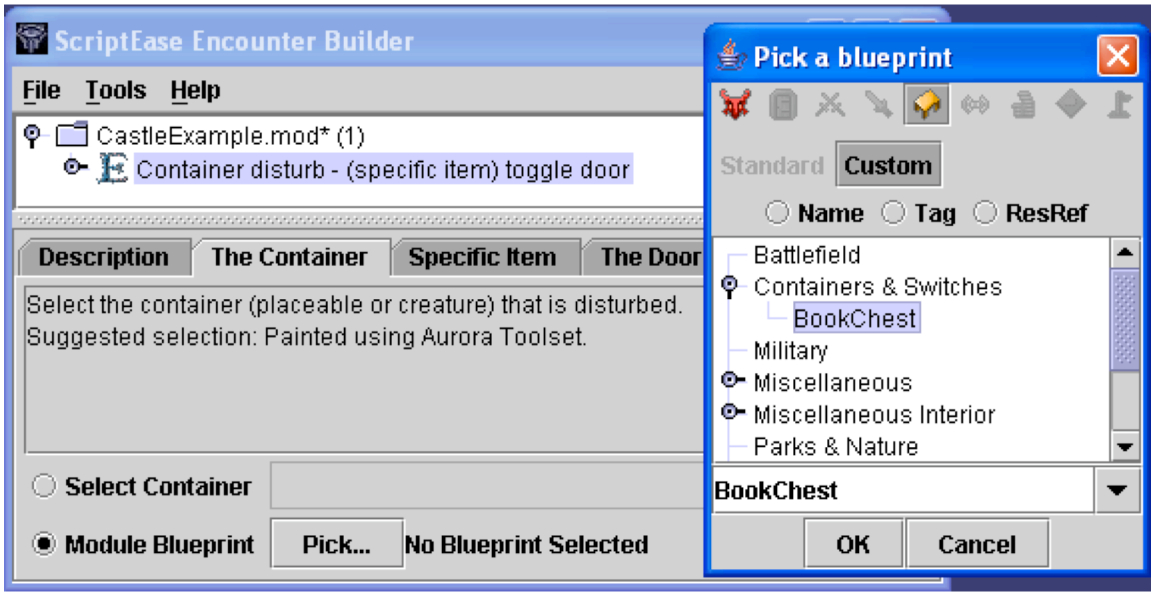

ScriptEase Encounter Builder
File Tools Help
CastleExample.mod* (0)
New Folder
New Specific Encounter
Delete
Paste
Triggers
Placeables
Containers
Conversations
Doors
Items
Transitions
Creatures
Blank template
Custom
Container disturb - spawn creature
Container disturb - (specific item)
Container disturb - (specific items) compose
Container disturb - (specific item) toggle door
Container open/dead
Container open/dead - create unique item
Container spell cast at - (specific spell, specific item)
Container use

Figure 6: Creating an encounter pattern.

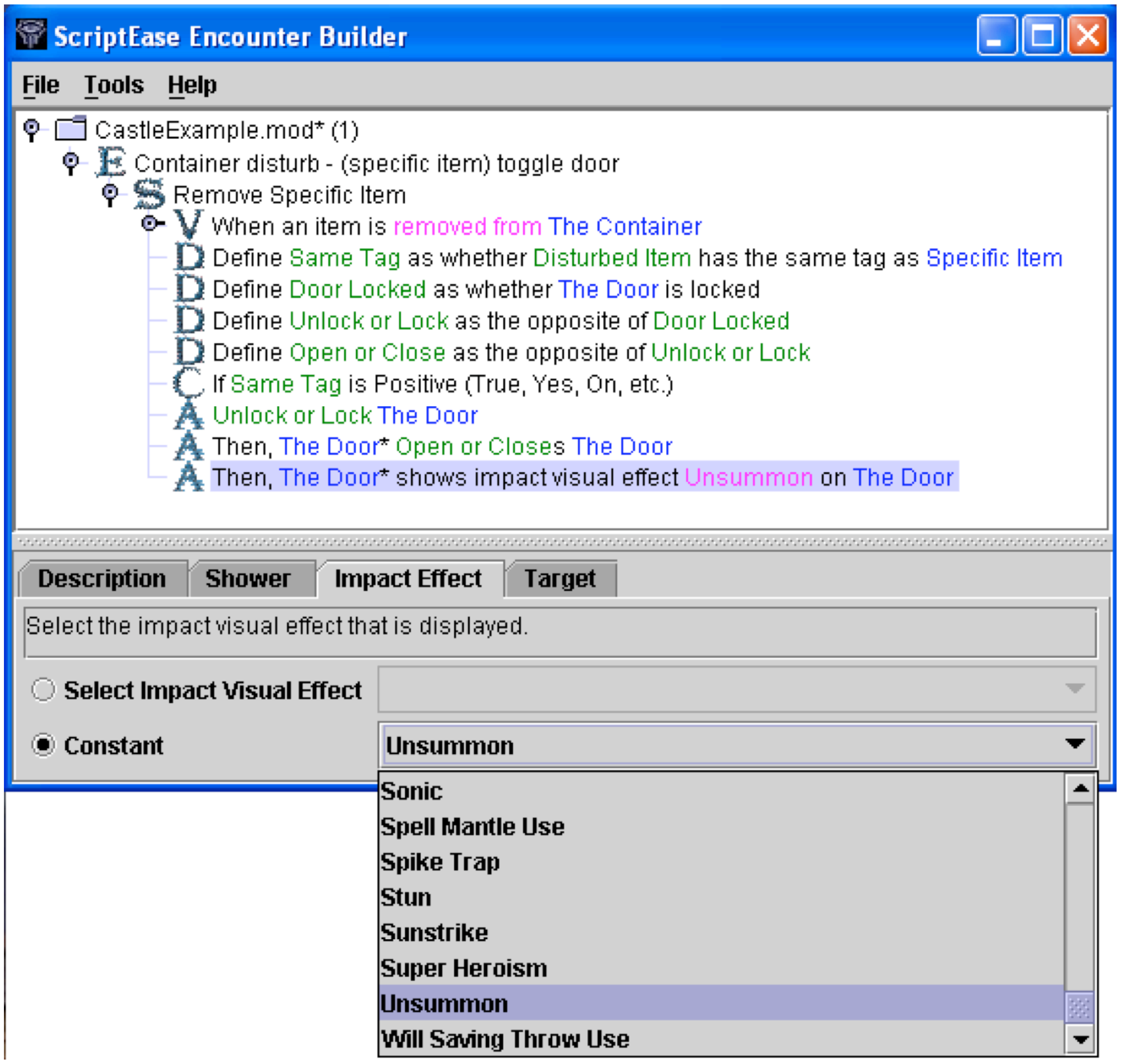


Select the container (placeable or creature) that is disturbed.
Suggested selection: Painted using Aurora Toolset.

Figure 7: Parameters for a pattern.

After picking the objects that will play the roles, the writer could simply save and compile the story and exit ScriptEase. However, it would be a better story if the reader (player) had some hint that removing the book in one room unlocked and opened the door in the other. Therefore, we show how the writer can adapt this pattern to this particular story by causing a visual effect to appear on the appropriate door. This visual effect will draw the attention of the reader to the door that was affected. Figure 8 shows how the writer adapts the pattern by first opening the *E* (Encounter pattern) and the *S* (Situation it contains) to reveal the *V* (eVents), *D* (Definitions), *C* (Conditions), and *A* (Actions) that comprise it. The writer then adds a new action to fire an impact visual effect (labeled *Unsummon*), by selecting this action from a menu, similar to the one used to add the encounter in Figure 6. This particular visual effect was chosen since it can be seen from far away. The only action the story writer has added is this last action. The other components of this pattern (Situation, Definitions, Condition and Actions) were revealed when the pattern was opened – indeed these other components (along with the roles) define the pattern.

After saving and compiling the story, the writer can “test-drive” the story by opening it in NWN. Figure 9 shows what happens when the PC (Serenity) removes the book from the chest. The basic graphics show the screen just before the book is removed and the insets show the parts of the screen that change, just as the book is dragged from the chest’s inventory to the PC’s inventory. The changes are shown as arrows from the two parts of the scene to the insets that show the change. The visual effect can be seen as white lines in the still picture, but this picture does not do justice to the actual game where the visual effect is a startling animation of billowing white light, accompanied by sound effects that draw attention to the door of interest.



CastleExample.mod* (1)
Container disturb - (specific item) toggle door
Remove Specific Item
When an item is removed from The Container
Define Same Tag as whether Disturbed Item has the same tag as Specific Item
Define Door Locked as whether The Door is locked
Define Unlock or Lock as the opposite of Door Locked
Define Open or Close as the opposite of Unlock or Lock
If Same Tag is Positive (True, Yes, On, etc.)
Unlock or Lock The Door
Then, The Door* Open or Closes The Door
Then, The Door* shows impact visual effect Unsummon on The Door

Select the impact visual effect that is displayed.

Select Impact Visual Effect
Constant: Unsummon

Sonic
Spell Mantle Use
Spike Trap
Stun
Sunstrike
Super Heroism
Unsummon
Will Saving Throw Use

Figure 8: Customizing a pattern.

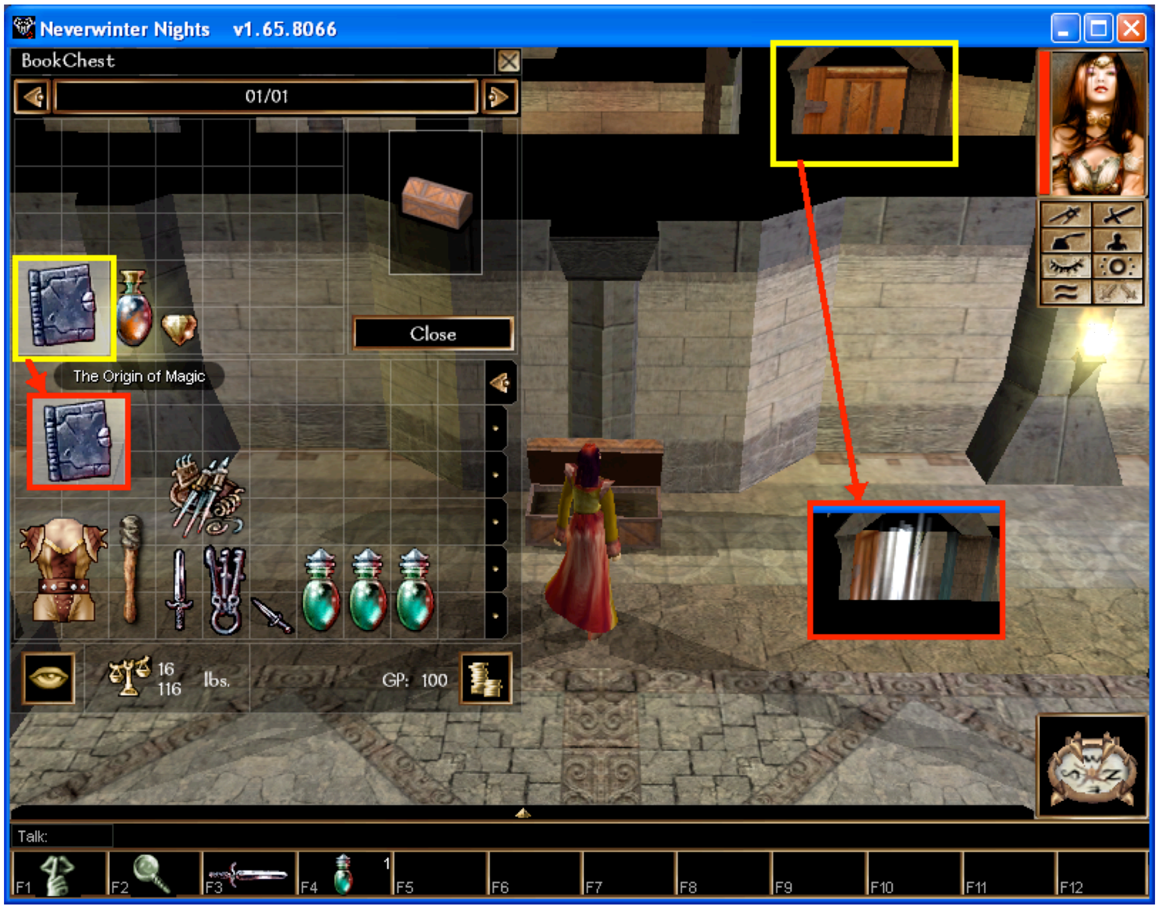


Figure 9: Playing the story.

The chest example shows the ease of using ScriptEase to create an interactive story. ScriptEase provides many benefits for simplifying the creation of interactive stories:

1. It provides a simple menu-driven interface.
2. All specifications are done using familiar story-element patterns.
3. The user need not know anything about NWScript.
4. ScriptEase organizes the (potentially) thousands of scripts.
5. Many common programming errors cannot happen. The patterns have been tested and debugged by the pattern designer, not the pattern user.
6. By enabling game designers to generate their own interactive stories, this eliminates the middleman (programmer) during story telling.
7. Creating interactive stories appeals to a wider audience of story writers, making it a useful tool for the classroom.

Claims of being "easy to use" are often made in the computing literature, rarely backed up with supporting evidence. For ScriptEase, we want to validate our claim that ScriptEase is easy to use by non-programmers.

INTERACTIVE STORY WRITING

Interactive story writing is a relatively new medium for creative expression. Compared to traditional pen-and-paper (word processor) stories, interactive stories are fundamentally different in their requirements and in the skill set needed.

It is often difficult for students to start writing a traditional short story and many students complain: “I don't know where to start”. In our experience, students have no difficulty starting an interactive story. Why? This fundamental difference is critical to understanding the pedagogical potential (and limitations) of interactive story writing.

To start a traditional story, it is necessary to set the scene of the story by writing a considerable amount of descriptive prose, before getting to the “good” part of telling the story. Thus students often limit their description of their setting, leaving the reader guessing and often confused. With an interactive story, the scene can be quickly “painted” using the Aurora Toolset to add scene components. We hypothesize three main consequences of this difference that we believe make it easier to start an interactive story (and therefore make it more fun).

1. Writing descriptive text can require more technical skill than selecting and placing scene objects. In an interactive story, the user can experience their imagination, both visually and aurally. They can create a mental image, and then select scenery, objects and sound to realize it. Pen-and-paper approaches require the user to translate images into words. The quality of the result depends on extensive education—in vocabulary, grammar and creative writing.

2. Feedback is slow and often difficult to evaluate in the traditional approach. It can take a long time to write the descriptive text and when the author reads what he/she has written, it is often hard to know if the text adequately describes the scene, or whether the author's mental vision is compensating for what is missing from the text. Often the text must be read several times to discover what is missing or is inconsistent with the mental vision. With an interactive story, the author can set the scene quickly and view it immediately. This results both in a higher chance of alignment with the mental vision and less doubt as to whether the scene can stand alone without relying on information from the mental vision known only to the author. From a Computing Science perspective, this is visual debugging at its finest.

3. Early positive feedback from correctly setting part of the scene increases self-confidence in technical skills. Steady obvious incremental improvement of the story results in an increase in motivation since the author is (usually) getting closer to the goal, rather than making progress at an irregular rate due to false starts.

Creating an interactive story is, in many ways, more similar to writing a play than writing a short story. In a play, the author concentrates on the plot, theme and character dialogue; other issues such as detailed scene descriptions, internal character subtleties, and flowing prose to tie it all together are not needed. An interactive story is similar. All the effort goes into creating the plot and composing the interactions between the characters. Scene descriptions are simply expressed as pictures, without the need for elegant prose. Character subtleties, if not expressed as text such as in conversations, are currently beyond the capabilities of games such as Neverwinter Nights.

One major difference between interactive stories and plays is in the background characters populating a scene. For example, if the hero of the story goes into a store to buy something, the play author just assumes that there are other people in the store and that there is a cashier. Little time has to be spent on the details, since the author (and the readers) can infer the rest. In an interactive story, however, the details of the store must be specified. All background characters (such as the cashier, and other NPCs) must be programmed. This extra level of detail may put a significant onus on the story creator, depending on the needs of their story. However, this is precisely where a tool such as ScriptEase can be a big win. Character behaviors can also be patterns. There could be, for example, a "cashier" behavior. The story writer could create a character in the story and assign the cashier pattern to it. This would allow the writer to add more realism to their story by easily populating a scene with additional characters exhibiting realistic behavior. Behavior patterns are the subject of current research in ScriptEase. In our pilot study, the students did not have access to any behavior patterns.

CLASSROOM PILOT

Working collaboratively with a high school English teacher and a high school student, a series of tutorials were created (for the tools Neverwinter Nights, Aurora, and ScriptEase) [9]. The high school teacher developed, and the high school student tested, an interactive story-writing assignment targeted for High School English students. This process took several months and several iterations of the documentation.

The interactive writing assignment was used as part of the curriculum in a Grade 10 English class, and administered over a two-week period in November 2004. It consisted of two components:

1. The students were taken on a field trip to the University of Alberta for the tutorials (because of the availability of the computing equipment). Over two days, they went through the Neverwinter Nights, Aurora and ScriptEase tutorials. Their high school teacher supervised the trip, and helped answer student questions. Two graduate students were on hand at all times to provide additional technical support. At the end of the trip, students were ready to create their stories.

2. Three eighty-minute English classes were completed in the high school computer lab, allowing the students to work on their stories. Extra computer hours were made available for those who wanted it.

Twenty-one students completed the assignment.

We found that the students were generally highly motivated to work on their stories. This became obvious early on when the students requested that additional computer lab time be made available for them to work on their stories. Part of the motivation likely came from the novelty of the classroom experience. Beyond that, however, there was a sense of excitement as the student's interactive story-writing capabilities increased.

In a regular English classroom, it is common for the teacher to have students "peer evaluate" each other's work. Students often interpret this as added work and are not very interested in what their peers have written since the stories are not professionally published. Therefore, the feedback is often minimal and little benefit to either student. During the field trip and upon return to the school we noticed a strong sense of collaboration. Students were neither encouraged nor discouraged from helping each other, exchanging story ideas, and providing feedback on other student's stories. In fact, we did not even think about interpersonal collaboration in designing the activity. What we discovered was that students began collaborating on their stories from the beginning of the tutorial exercises without encouragement from their English teacher. One student would spontaneously say to the next student, “look what I tried” and the other student would immediately get involved by adapting the idea to his/her own story or by suggesting related things to try. Groups of students began gathering at one workstation or another observing particular students' activities. This encouraged all the students to produce better stories, knowing that their work was being seen and appreciated by their fellow students. Constructive collaboration within a community of learners provided students with an opportunity to improve their critiquing skills. We believe that it also resulted in better understanding of concepts since students would often try to explain things (that they had figured out by themselves or that were clarified by the teacher) to their peers. In some cases, the interaction went beyond merely giving instructions; it became interactive collaborative story development. From the Computing Science perspective, it appeared that some students had discovered the advantages of pair programming.

We were only able to observe the collaboration during the two-day trip to the University of Alberta. However, the collaborations continued throughout the assignment. The teacher contrasted this high level of interpersonal collaboration in interactive story writing with the collaboration in traditional learning activities:

> “ScriptEase created interaction among my students which was not typical of an individual activity such as story writing. This way of story writing encouraged collaboration before, during and after their stories were complete.”

The teacher gave us additional insights into the student’s experience. Of particular interest is the observation that some students with lower academic achievements became immersed in the assignment. A possible explanation is that the creative mechanism of interactive stories was easier for them or better suited to their capabilities. If so, this has exciting implications for the pedagogical development of the creative “writing” courses. Gardner's work on different modes of intelligence and their connection to expressive modes (music, dance, etc.) seems to apply here, since our interactive stories contain visual and sound aspects [2].

The teacher reported that the students had a high level of curiosity about the capabilities of the interactive story-writing tools. Many students asked questions about tool features beyond what was introduced for their assignment, and some students took this one step forward by exploring these features even though no documentation was provided.

In general, everything went well with the pilot study. There were a few glitches, such as some computer hardware problems and some (small) issues with the tutorials. None of these detracted from the student’s enjoyment of the experience in any significant way.

The one big mistake that we made in the pilot study was underestimating the time it would take the teacher to mark the assignments. The teacher developed a rubric to mark the story. However, each component of the grading required the teacher to play-test the student submission. The result was that each story required several hours of grading time. Not knowing the details of a student’s story, the teacher had to explore the student’s game world to uncover all the subtleties of the story (for example, traps, hidden doors, and dead end passageways). For subsequent classroom use, we will ask the students to create storyboards for their interactive story. By having the students map out the major components of the story, the teacher can avoid most of the trial and error exploration aspects of the game. This is often done traditionally in the classroom through a written outline of the story provided by the student.

This pilot has been tremendously valuable for giving feedback on the use of interactive storytelling in general, the computer tools in particular, and ways to improve the tutorials, grading scheme, student-computing environment, and scope of the assignment. The initial experience was very positive for all parties. In particular, the teacher conducted a student survey after the assignment was completed and reported a very high level of satisfaction from the respondents. Most students gave a strong preference to doing another interactive story-writing assignment over a traditional story-writing assignment.

FUTURE WORK

An experimental study using ScriptEase will happen in May 2005 using another English class as the target population. We will gather data on the students and their performance. Data will be used to identify any correlations between student abilities (e.g. problem solving skills) and background (e.g. computer experience), and how well they do on the assignment. Further, we hypothesize that some students who have difficulty expressing their creativity in words using traditional technologies may have no such limitations using interactive story-writing technology. On the other hand, there may be other students who excel at traditional writing, but that do not have sufficiently developed logical thinking skills to design and create an interactive non-linear story.

The development of interactive story-writing technology is still in its early stages. ScriptEase has been two years in the making, and there is still much to do. Our current work is building behavior patterns, allowing the user to rapidly create scenes populated with a rich set of NPCs. Future work will include conversation patterns. Currently, creating conversations is cumbersome and time consuming, yet many types of conversations are really just frequently occurring patterns (e.g., a greeting, or a request for a service). Enhancing ScriptEase to include conversation patterns is a natural extension of our current work.

Our research goal is to make interactive story-writing technology available to non-programmers, demonstrate its pedagogical value in the classroom, and work towards popularizing this medium as a new form of creative literature.

ACKNOWLEDGEMENTS

This research was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC), the Institute for Robotics and Intelligent Systems (IRIS), and Alberta's Informatics Circle of Research Excellence (iCORE).

REFERENCES

1. BioWare. http://www.bioware.com.
2. Gardner, H. Multiple Intelligences: The Theory in Practice. Basic Books, New York, 1993.
3. McFarlane, A., Sparrowhawk, A., and Heald Y. "Report on the Educational Use of Games," in Teachers Evaluating Educational Multimedia Report, 2002. http://www.teem.org.uk/publications/teem_gamesined_full.pdf.
4. Lilac Soul. http://lilacsoul.revility.com.
5. McNaughton, M., Cutumisu, M., Szafron, D., Schaeffer, J., Redford, J., and Parker, D. "ScriptEase: Generative Design Patterns for Computer Role-Playing Games," in 19th International Conference on Automated Software Engineering, pp. 88-99, 2004.
6. Neverwinter Nights Vault. http://nwvault.ign.com.
7. Robertson, J. and Good, J. "Story Creation in Virtual Game Worlds," in *Communications of the ACM*, vol. 48, no.

1, pp. 61-65, 2005.
8. ScriptEase project. http://www.cs.ualberta.ca/~script.
9. Tutorials and assignment available at http://www.cs.ualberta.ca/~script/scripteasenwn.html.