A Reusable Scripting Engine for Automating Cinematics and Cut-Scenes in Video Games

M. McLaughlin and M. Katchabaw
The University of Western Ontario
mmclaug3@uwo.ca, katchab@csd.uwo.ca

Abstract

Storytelling can play a critical role in the success of modern video games. Unfortunately, it can often be quite difficult for storytellers to directly craft content for games, typically requiring them to work with programmers to implement story elements. This needlessly complicates the development process, straining scarce resources while potentially hampering creativity and story quality at the same time. As a result, supports and tools are necessary to enable storytellers to generate story content for games directly, with minimal programming or programmer assistance required, if any.

This paper introduces a Reusable Scripting Engine to automate the generation of cinematics and cut-scenes in video games. This approach allows storytellers to provide their stories in a well-defined, structured format, which is then interpreted by our engine, along with supplemental graphic and audio content, to produce an animated presentation of the story in an automated fashion. This paper presents the design of our Reusable Scripting Engine, and discusses a prototype implementation of this design, as well as initial experiences with using this prototype system to date.

Introduction

Storytelling is recognized as an important element of modern video games (Bateman, 2007; Glassner, 2004; Krawczyk & Novak, 2006), and can often make or break the success of a game. For games dependent on their story elements, some say that the need for quality in storytelling at least equals the need for quality in graphics, sound, and general programming of the game (Sheldon, 2004). Consequently, it is critically important to provide the necessary supports to allow storytellers to ply their craft within games. Doing so, unfortunately, poses several challenging problems.

The current state-of-the-art in storytelling in games typically requires some form of programming background or expertise, requiring the writing of code to implement story elements, which is typically outside the realm of most storytellers (Carbonaro et al., 2005). Similarly, most game programmers lack the necessary writing skills and background for effective storytelling (Koster, 2005; Krawczyk & Novak, 2006). Consequently, to support storytelling in games, storytellers must rely on programmers to implement their stories, which

Proceedings of CGSA 2006 Symposium

can be expensive and time consuming, with the potential for communication errors, programming errors, and a loss of direct and total control affecting the story creation process. In independent or amateur development efforts, programming skill might not be readily available, confining storytellers to simple programming or copying and changing existing programs, which greatly hampers the creative process and overall story quality as a result (Onuczko et al., 2005).

To assist storytellers, tools and supports are necessary to allow them to tell their stories in games with minimal programming or programmer support required, if any. Such tools would need to allow storytellers to convey their stories in natural language, graphically, or in some other simple form, while automation prepares this story content for use with little or no human intervention required. Unfortunately, work in this area is quite scarce. One notable exception is ScriptEase (Cutumisu et al., 2005), an innovative pattern and template-driven approach, primarily aimed at in-game storytelling and behaviour control of non-player characters. Interesting work also comes in the form of Bubble Dialogue (Cunningham et al., 1992), developed primarily as a tool to investigate communication and social skills, particularly in educational settings. Bubble Dialogue, however, is intended to be a stand-alone tool not suitable for embedded use in video games, and it is questionable whether its interface, designed for novices to easily construct stories, would be expressive, flexible, and powerful enough for professional storytellers. Other related work is an interesting game entitled The Movies (Lionhead Studios, 2005). While this game allows players to construct their own stories for their own films, the general approach and interface might not be the most productive or easiest one for storytellers to use in crafting stories for use in other games.

Aside from in-game storytelling embedded in gameplay, cinematics and cut-scenes are two of the more common techniques for storytelling in games, conveying story through visuals and audio, typically presented much like a dramatic piece (Krawczyk & Novak, 2006). This paper presents a modular and reusable scripting engine for automating the creation of these storytelling elements, providing an easy to use tool driven by storytellers and content producers, without requiring any programming whatsoever. As input, storytellers provide their stories formatted according to Text Encoding Initiative (TEI) guidelines for dramatic pieces, which use Extensible Markup Language (XML) to provide a precise and formal representation of their work (Sperberg-McQueen & Burnard, 2004). Several excellent tools exist to craft or convert stories to TEI Guidelines (The TEI Consortium, 2006). Our software engine processes these stories, along with supplemental graphical and audio content, to present and animate a cinematic or cut-scene based on the stories. Since the stories are encoded in a formal representation, the interpretation required to do this is a straightforward process.

This paper provides details of the design and prototyping of our engine, called the Reusable Scripting Engine, developed using the OGRE 3D rendering engine (The OGRE Team, 2006), a popular engine used in many independent, hobbyist, and academic game projects. This paper also discusses our experiences through using it to automate the creation of cinematics and cut-scenes, providing examples of story scripts and results from our engine. To date, results have been quite positive and are very promising for continued work in this area in the future.

The remainder of this paper is organized as follows. We begin by introducing and discussing the fundamental concepts behind story scripting and scripting languages. We then describe the design of our reusable scripting engine for automating cinematics and cut-scenes in

video games and examine a prototype implementation of our engine based on this design, and present initial results from using this prototype system to date. Finally, we conclude this paper with a summary and a discussion of directions for future work.

Story Scripting and Scripting Languages

To automate the presentation of story elements from a story in a video game, it is necessary for the original story to be scripted in such a way that it can be easily acted out. Consequently, this scripting must identify characters, dialogue, voice directions, stage directions, setting, and other elements common to traditional dramatic pieces. Fortunately, storytellers must provide this information for cinematics and cut-scenes for traditional storytelling techniques in games currently in use (Bateman, 2007), so the need for this information is not a new imposition created by the automation process.

A difference, however, comes in the form of the rigor with which this information must be provided for automation to be effective. Scripting provided for automation will need to be precise and formal enough to be easily processed and understood by the software automating the presentation of the story. Otherwise, there may be ambiguity and confusion in the script, resulting in decreased quality of the overall story experience created. Consequently, there is a need to provide a structure and standardized approach to scripting for storytelling within video games for automation efforts to be successful.

Fortunately, the Text Encoding Initiative (TEI) has developed an XML-based specification for marking up various kinds of texts, including dramatic pieces (Sperberg-McQueen & Burnard, 2004). TEI guidelines provide an extensive set of tags for structuring dramatic pieces and identifying all of the elements listed above that must be defined for cinematics and cut-scenes in video games. Consequently, the TEI guidelines for dramatic pieces provide an excellent starting point for adding precision and formality to script specifications for automated presentation in video games.

Unfortunately, XML is not the most natural platform for storytellers writing stories, and requiring storytellers to produce stories with embedded TEI tags needlessly complicates the story creation process. To assist in the process of working with TEI tags, there are numerous software packages available that adhere to TEI guidelines for importing existing works or writing them from scratch (The TEI Consortium, 2006). Several of these packages plug into existing word processing or office productivity software, or otherwise work with this software, to ensure that storytellers need not give up their favourite story writing tools to be able to take advantage of the TEI guidelines. This can greatly facilitate the story creation process, particularly when it comes to automation.

In the end, however, we could not completely follow TEI guidelines in our current work, and instead had to derive tailored guidelines for automation purposes, with some modifications and extensions to the existing TEI guidelines. This was necessary for several reasons. First, TEI guidelines are incredibly detailed and require information that does not quite make sense or fit well within the realm of cinematics or cut-scenes for video games. Second, several elements in the TEI guidelines are not formal or precise enough yet for our purposes. For example, stage directions in the TEI guidelines are too open and too flexible; thus additional structure and

formality needed to be applied to ensure these directions could be followed automatically. Third, we needed additional elements to link game content and assets into story scripts. For example, character models, scene backgrounds, and other art assets must be identified and linked with corresponding elements within scripts, as well as audio assets for background music, voice-overs for dialogue, and so on. Finally, other new elements were necessary to make managing story content within a game easier. For example, dialogue elements needed to be split into more manageable chunks that could be displayed on-screen and linked with voice-overs as necessary.

The end result of this work was a set of customized guidelines for representing story elements for automation within video games, based on the TEI guidelines. A complete specification of the customized version can be found in (McLaughlin, 2006). For now, it is simpler to present examples demonstrating their application to a story sequence, in this case from the movie The Princess Bride (Goldman & Reiner, 1987). Figure 1 provides a segment from the header of a scene from this movie, defining the cast list, and various other elements that are to be used in performing this scene. When defining a character, we provided a name, an identifier for the character, as well as several models to be used to represent the character throughout the scene. Figure 2 presents a segment from the body of the scene, showing various lines of spoken dialogue as well as stage directions to be used in acting out the scene.

```
<performance title="Battle of Wits">
  <!--Information about the graphical elements of the play-->
  <header>
    <!--Cast of characters and props-->
    <castList>
      <character id="vizzini">
        <name>
          Vizzini
        </name>
        <!--Character models defined for this character.  The first model is designated the default model-->
        <availModelList>
          <model id="defaultViz" location="c:\performances\bow\models\vizzini\default.bmp"/>
          <model id="vizziniDrink" location="c:\performances\bow\models\vizzini\drinking.bmp"/>
          <model id="vizziniLaugh" location="c:\performances\bow\models\vizzini\laughing.bmp"/>
          <model id="vizziniDead" location="c:\performances\bow\models\vizzini\dead.bmp"/>
        </availModelList>
      </character>
            …
  </castList>
  …
</header>
```

Figure 1: Header Elements from a Sample Story Script

```
…
<dialogue speaker="vizzini">
  <line>Well, I- I could have sworn I saw something.</line>
  <line>No matter.  First, let's drink.</line>
  <line>Me from my glass, and you from yours.</line>
</dialogue>
<stageDirection>
  <costumeChange characterID="vizzini" model="vizziniDrink"/>
  <costumeChange characterID="westley" model="westleyDrink"/>
  <pause duration="3"/>
  <costumeChange characterID="vizzini" model="defaultViz"/>
  <costumeChange characterID="westley" model="defaultWest"/>
</stageDirection>
<dialogue speaker="westley">
 <line>You guessed wrong. </line>
</dialogue>
<stageDirection>
  <costumeChange characterID="vizzini" model="vizziniLaugh"/>
</stageDirection>
<dialogue speaker="vizzini">
 <line>You only think I guessed wrong! That's what's so funny!</line>
 <line>I switched glasses when your back was turned!</line>
</dialogue>
...
```

Figure 2: Segment of the Body from a Sample Story Script

Again, it is important to note that a storyteller would not need to create this story with the required tags already in place. Authoring tools can simplify the process as discussed above, making the markup process no more difficult than creating a regular document or web page. With these tags and annotations in place, the story script is ready to be presented in an automated fashion within a video game using our Reusable Scripting Engine, as discussed in detail in the next section.

Reusable Scripting Engine Design

With story scripts for cinematics and cut-scenes in place, this section examines the design of a Reusable Scripting Engine capable of processing these scripts and automatically presenting them within a video game. We begin by presenting the architecture of our engine in Figure 3, and then proceed to discuss the major modules of this architecture in the subsections that follow. We conclude this section with a discussion of the interactions between the modules in a sample automation scenario.

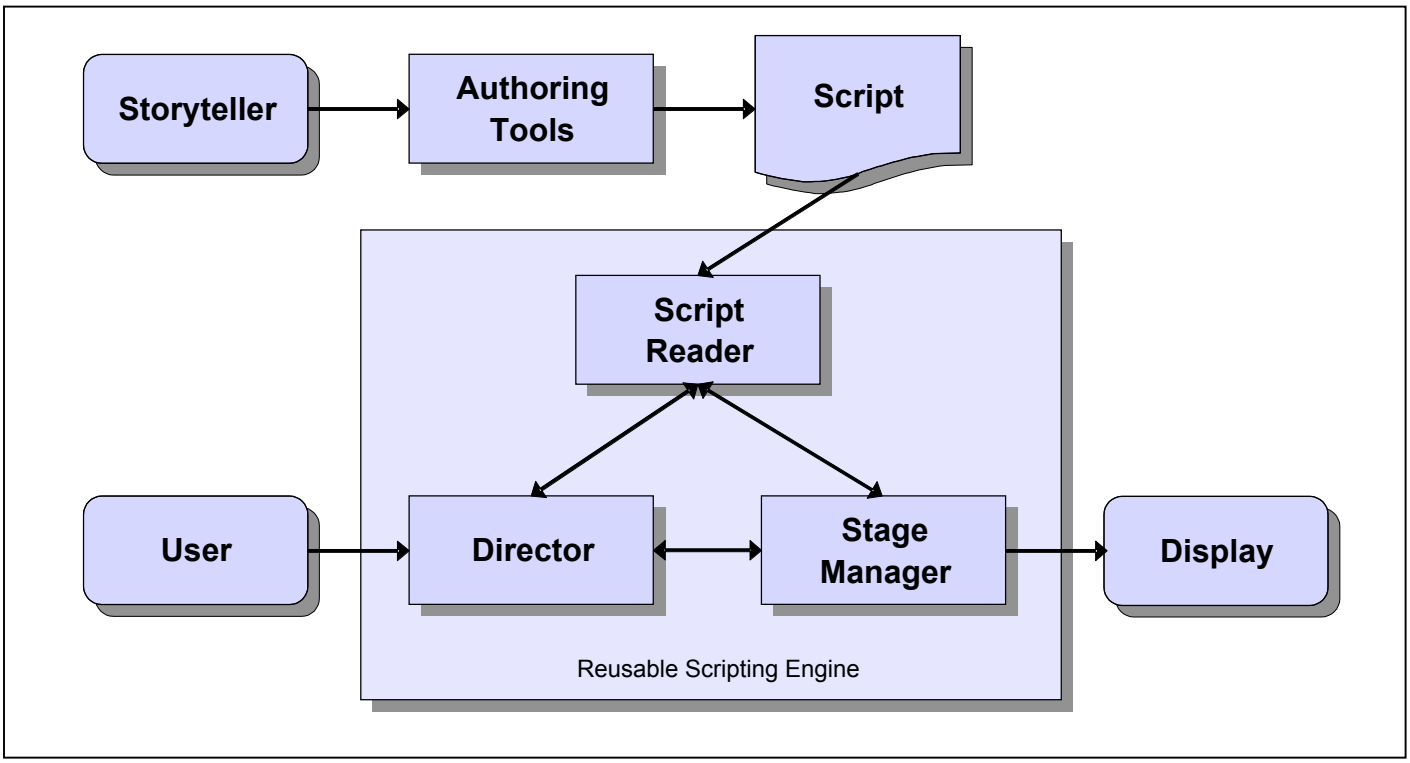

Figure 3: Reusable Scripting Engine Architecture

Director

The primary role of the Director in the engine is to manage the Script Reader and Stage Manager modules to oversee the entire production and presentation of the cinematic or cut-scene. As such, it handles internal object management and communication tasks as required for the engine. The Director module is also responsible for managing any interactions with the user of the engine, which, depending on the context, could either be the player of the game in question or the game itself. These interactions could include playback control to regulate the flow of the cinematic or cut-scene, as well as any other access required to the engine.

Script Reader

As the name implies, the Script Reader module reads in the story script and processes it to prepare it for use in the engine. This requires the module to parse the XML representation of the script to find the major elements of the story, verify the correctness and completeness of the script, and fill in any missing or assumed elements of the story where possible. When the script is deemed ready for performance, the Script Reader generates a collection of stage actions from the script, creating a performance, and passes this performance on to the Director module to have the performance executed.

Stage Manager

The Stage Manager module is responsible for generating the actual on-screen performance of the story script read in by the Script Reader module. This module receives its direction on what to do, how to do it, and when to do it from the Director module, which is basing its directions on the collection of stage actions generated by the Script Reader. The Stage Manager also reports back to the Director on the status of the production as it progresses. Any commands or directions received from the Director are executed immediately, to provide the Director a good measure of control over how the story is presented.

The Reusable Scripting Engine in Action

Having examined the various components of the architecture for the Reusable Scripting Engine, we can now look at its operation in more detail in processing and presenting a story script automatically for a video game to produce a cinematic or cut-scene. Generally, this would be comprised of the following steps:

1. The storyteller begins the process by using one of the various authoring tools mentioned in the previous section to create a story script, with the appropriate embedded tags and annotations necessary for it to be processed by the engine.

2. The game that uses this particular story script decides that it needs the script to be acted out on-screen. As a user of the engine, it notifies the Director module to prepare the script for presentation.

3. The Director calls upon the Script Reader module to load and process the script. The script reader does so, ensuring the script is correct, complete, and ready for presentation. Based on this, Script Reader provides a collection of stage actions for the performance back to the Director.

4. If the game required that the script be performed immediately, the Director proceeds. Otherwise, the Director waits until it is instructed by the game to begin the performance. This way, scripts can be pre-loaded and prepared for presentation before they are actually needed.

5. To have the script performed, the Director passes stage actions from the collection it received from the Script Reader to the Stage Manager module. The Stage Manager module follows these actions, causing the script to be performed on the display for the game.

6. At any point during the performance, the game can issue commands to control the playback of the cinematic or cut-scene, either because it deems them necessary, or in response to player input. These commands are received and processed by the Director, which instructs the Stage Manager accordingly.

7. The Stage Manager keeps the Director module informed about the status of the cinematic or cut-scene throughout its performance. When the performance is complete, the Stage

Manager notifies the Director, which in turn notifies the game, to allow the game to proceed from that point.

8. The script, stage actions, and other resources used in the performance are not disposed of until the game instructs the Director explicitly to do so. (This is done in case a repeat of the performance is required.) Once instructed to do so, the Director disposes of these elements, freeing resources for other performances.

Prototype Implementation

Based on the Reusable Scripting Engine design from the previous section, a prototype has been implemented for Microsoft Windows XP, written in C++ in Microsoft Visual Studio .Net. To support script reading, XML processing was implemented using Microsoft's XML libraries as they tend to be among the more robust and easy to use XML facilities for this environment.

For visual rendering of the cinematic or cut-scene, we used the OGRE 3D rendering engine (The OGRE Team 2006). OGRE 3D is a popular engine used in many independent, hobbyist, and academic game projects. While not as robust as commercial engines available, OGRE 3D is free and open source, and provides both 2D and 3D support to our Reusable Scripting Engine. Furthermore, there are several other on-going game research and development projects at the University of Western Ontario that are currently making use of OGRE 3D; using OGRE 3D as a foundation in this work allows our Reusable Scripting Engine to more easily integrate with these other projects in the future.

For audio support, we are using the FMOD package (The FMOD Team, 2006). Again, this was chosen because it is free and also to allow easy integration with other on-going projects.

Results to Date

To date, we have begun to stage some simple performance using the prototype implementation of our Reusable Scripting Engine. The performances are not yet flawless, and still have the odd glitch here and there, largely due to various on-going issues with the OGRE 3D rendering engine. In general, the prototype works well, and development is continuing.

Integration into several of the other game research and development projects under way at the University of Western Ontario is currently in progress. Results are very encouraging, as the use of the engine allows researchers and developers on those projects to focus on the story elements of their projects, without needing to worry about implementation details as to how the stories will be told and presented in their games. Feedback to date has been quite positive.

To demonstrate the Reusable Scripting Engine, we have provided a sequence of images from a re-enactment of a portion of an episode of The Simpsons (Stern, 1993), "Duffless". In this particular scene, the set is the living room of the Simpson's home (in this case, the image is of a real-life recreation of the Simpson's home in Las Vegas).

Figure 4a shows the standard graphical elements that appear when displaying a performance. The white text box at the bottom is where all communications will take place and will only be visible when dialogue is spoken. Whenever there is a pause there is no text box

visible. At the beginning of each dialogue element the name of the character speaking the line appears.

Figure 4: Scenes from the Re-enactment of The Simpsons Episode "Duffless"

Figure 4b shows two important features: the ability to change character models while the performance is in progress, and changes to the text box based upon the character's tone (in this

case, Homer is scheming to himself and his dialogue is in the "thought" tone). It is also useful to note that even though this section of text is part of a new dialogue element in the script (required since there is a change in tone), the speaker's name does not appear at the beginning of the text. Character's names will only appear at the first line of dialogue by a new speaker or the first line of dialogue in a scene. Figures 4c and 4d demonstrate these elements as well.

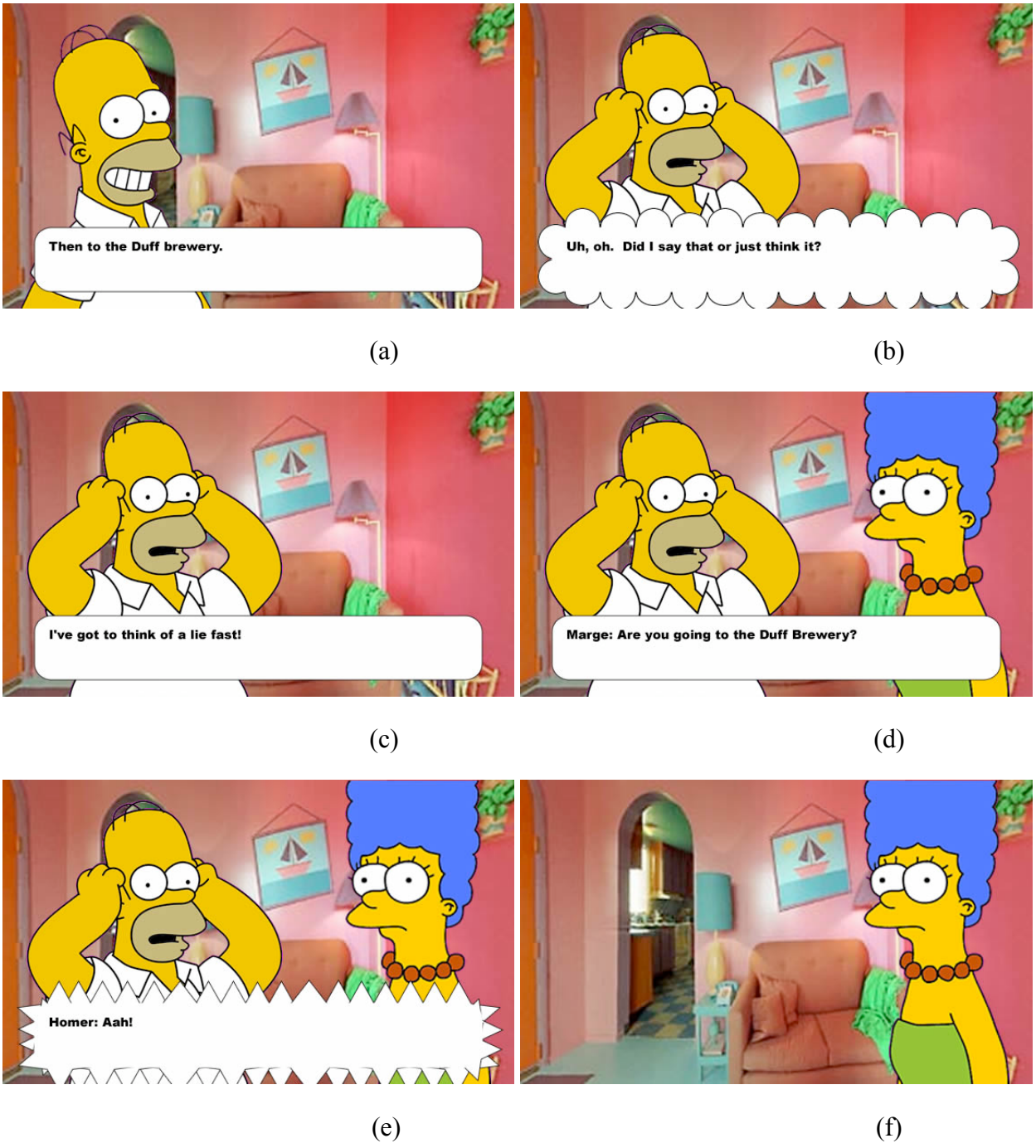

Figure 5: More Scenes from the Re-enactment of The Simpsons Episode "Duffless"

Figure 4e shows what the player sees during a pause in the action. For the duration of the timed pause, the scene remains static. Alternatively, a game using this engine could pause the playback through the Director module, but this would require the game to closely monitor the status of the performance. In this case, the pause is important to show that Homer's brain and Homer's mouth are completely distinct entities, and get out of sync with one another. This process begins in Figure 4f, when Homer says what he should be thinking and vice-versa, and continues through Figure 5a, 5b, and 5c, with Homer getting visibly alarmed as he realizes what has transpired through another character model change.

It is impossible to show the animation that would occur in Figure 5d, but Marge would slide in from the right side of the stage according to her stage direction. Also note in this scene that the dialogue is preceded by the new speaker's name. This occurs whenever a section of dialogue for a new speaker is encountered. Figure 5e shows how the third voice tone, "shout", is displayed on the screen, when Homer realizes that his *faux pas* has been noticed by Marge and the rest of the family. After accidentally revealing his plans, Homer flees out stage left in Figure 5f. After a short pause, Marge exits stage right and the scene concludes at this point.

Conclusions and Future Work

Storytelling has been shown to be a very important element in modern video games: in many cases the success or failure of games depends on their story elements. Consequently, tools and supports are necessary to enable storytellers to directly produce story content for games, without requiring programming background and expertise. This will allow stories for video games to be crafted more efficiently and more effectively, easing the development process and potentially increasing the quality of the games as a result.

Our Reusable Scripting Engine provides a step in this direction for cinematics and cut-scenes in video games, by giving storytellers the ability to directly input their story content into a game and have it automatically performed as a result. Results in using our prototype engine to date have been quite positive, and demonstrate great promise for the future. Potential directions for continued work in this area include the following:

- Providing additional support for audio, including spoken dialogue, background music that changes, and sound effects as necessary. Work with FMOD for this is currently on-going.

- Support for animated characters and background elements and effects is also important. Static characters with little or no animation are still in use in many games today, but our engine must support more than that in the future.

- Support for 3D cinematics and cut-scenes is also necessary, and is fortunately possible within the OGRE 3D engine. This will require the addition of new stage directions to our current scripting capabilities to work in a truly 3D space.

- There is currently considerable interest in dynamically generated story elements in video games. The Reusable Scripting Engine should be extended to support these efforts.

- Support for in-game storytelling sequences in video games is also necessary, in addition to the current focus on cinematics and cut-scenes.

References

Bateman, C. (Ed.). (2007). *Game writing: Narrative skills for videogames.* Boston, MA: Charles River Media.

Carbonaro, M., Cutumisu, M., McNaughton, M., Onuczko, C., Roy, T., Schaeffer, J., et al. (2005, June). *Interactive story writing in the classroom: Using computer games*. Paper presented at the 2005 International Digital Games Research Conference, Vancouver, BC.

Cunningham, D., McMahon, H., & O'Neill, B. (1992). Bubble dialogue: A new tool for instruction and assessment. *Educational Technology Research and Development*, 40(2), 59-67.

Cutumisu, M., McNaughton, M., Szafron, D., Roy, T., Onuczko, C., Schaeffer, J., et al. (2005, June). *ScriptEase - A demonstration of ambient behavior generation for computer role-playing games*. Demonstration presented at the First Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE), Marina Del Rey, California.

FMOD Team, The. (2006). The FMOD sound system [Software]. Retrieved January 19, 2007, from http://www.fmod.org.

Glassner, A. (2004). *Interactive storytelling: Techniques for 21st century fiction*. Natick, MA: A K Peters Limited.

Goldman, W. (Writer), & Reiner, R. (Director). (1987). *The princess bride* [Motion Picture]. USA: 20th Century Fox.

Koster, R. (2005). *A theory of fun for game design*. Scottsdale, AZ: Paraglyph Press.

Krawczyk, M. & Novak, J. (2006). *Game development essentials: Game story and character development*. Clifton Park, NY: Thomson Delmar Learning.

Lionhead Studios. (2005). The Movies [PC Game]. Activision.

McLaughlin, M. (2006). *Reusable storytelling engines*. Unpublished undergraduate thesis, The University of Western Ontario, London, Ontario, Canada.

The OGRE Team. (2006). OGRE manual v1.2.4 ('Dagon') [Software Manual]. Retrieved January 20, 2007 from http://www.ogre3d.org/docs/manual.

Onuczko, C., Cutumisu, M., Szafron, D., Schaeffer, J., McNaughton, M., Roy, T., et al. (2005, August). *A pattern catalog for computer role playing games*. Paper presented at Game-On North America 2005, Montreal, Canada.

Sheldon, L. (2004). *Character development and storytelling for games*. Boston, MA: Thomson Course Technology.

Sperberg-McQueen, C. & Burnard, L. (Eds). (2004). *Guidelines for text encoding and interchange: XML-compatible edition* [Published for the TEI Consortium]. Oxford: Humanities Computing Unit, University of Oxford.

Stern, D. (Writer) & Reardon, J. (Director). (1993). Duffless. [Television series episode]. In *The Simpsons* [Episode 9F14]. Los Angeles, CA: 20th Century Fox Broadcasting Company.

The TEI Consortium. (2006). TEI Software [Computer software]. Charlottesville, VA: Insititute for Advanced Technology in the Humanities, University of Virginia. [Available at: http://www.tei-c.org/Software/].