Story Scripting for Automating Cinematics and Cut-Scenes in Video Games

W. Zhang, M. McLaughlin, and M. Katchabaw
Department of Computer Science
The University of Western Ontario
London, Ontario, Canada
N6A 5B7
wzhang95@csd.uwo.ca, mmclaug3@uwo.ca, katchab@csd.uwo.ca

ABSTRACT

Storytelling can play a very important role in the success of modern video games. Unfortunately, it can be quite difficult for writers to directly create and integrate story content into games on their own, and they must instead rely upon programmers and others on the development team to implement their stories. This needlessly complicates the game development process, leading to increased costs, more strain on developer time, and loss of creative control and, potentially, story quality as a result. Consequently, tools and supports are necessary to enable writers to generate story content for games directly, with minimal programming or programmer assistance required, if any.

This paper examines the use of specialized story scripting elements to automate the production of cinematics and cut-scenes for video games. These elements allow writers to specify their stories in a well-defined, structured format that can be acted out automatically by software. This paper discusses these story scripting elements in depth, along with a prototype software engine capable of using these elements for cinematic and cut-scene automation. This paper also presents experiences with using this engine to recreate cinematics and cut-scenes from existing commercial video games.

Categories and Subject Descriptors

H.5.1 [**Information Interfaces and Presentation**]: Multimedia Information Systems – animations, artificial, augmented, and virtual realities; K.8.0 [**Personal Computing**]: General – games

General Terms

Design, Human Factors, Languages.

Keywords

Storytelling, automation, story scripting, cut-scenes, cinematics, video games

1. INTRODUCTION

Storytelling is widely recognized as an important element of modern video games [2,9,12], and in some cases is regarded as one of their most defining aspects [7]. Some predict that storytelling in games will continue to grow in prominence as new hardware and technologies create more opportunities for stories in games, and shifts in player audiences increase the demand for the inclusion of quality stories in games [4].

Delivering story in games, unfortunately, can be a challenging task. While story creation is naturally the responsibility of writers on the development team [2,17], these writers traditionally must work with programmers and others on the team to integrate story content into the game being developed due to the complexity involved and programming expertise required. This, however, can be expensive in terms of budget and scheduling resources [6], which is problematic considering the limitations often in place in creating story content for games [2]. Furthermore, this introduces a gap between storyteller and story [6], which can impact the creative process and overall story quality as a result. Consequently, a simpler, more streamlined story creation process for games is necessary—a need recognized for some time by industry practitioners [2,6].

Automating storytelling can alleviate these issues by allowing writers to tell their stories in games with minimal programming or programmer support required, if any. Following this approach, tools and supports would allow writers 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 relatively scarce, as discussed in Section 2 of this paper, and many outstanding problems remain unresolved as a result.

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 [12]. Our current work is a continuation of our earlier work in this area towards the development of a Reusable Scripting Engine designed specifically for automating cinematics and cut-scenes in games [16].

Our earlier work primarily focused on the core elements of the Reusable Scripting Engine, establishing an architecture and workflow that took story content from authoring tools through to

presentation. While this allowed basic cinematics and cut-scenes, this did not enable robust presentations with the high level of production values necessary for most modern video games. Consequently, our current work has focused on providing enriched story scripting elements, giving writers considerably more power, flexibility and expressiveness in story creation than our earlier work.

This paper introduces and discusses the details of our enhanced story scripting elements for video games, as well as their integration into our Reusable Scripting Engine platform. This paper also describes our experiences through using these enhanced elements to replicate cinematics and cut-scenes from commercial games, thereby demonstrating their effectiveness and suitability in automating the story creation process for video games.

The remainder of this paper is organized as follows. Section 2 presents related work in this area, both from research and industrial perspectives. Section 3 discusses story scripting for video games in general, and describes the scripting elements and approach taken in our current work. Section 4 presents the design and implementation of our proof of concept system, the Reusable Scripting Engine. Section 5 discusses our experiences from using this prototype, in particular presenting a case study in replicating cinematics and cut-scenes from commercial video games. Finally, in Section 6, we conclude this paper with a summary and a discussion of directions for future work.

2. RELATED WORK

This section discusses relevant related work, both from research and industrial perspectives. As mentioned in the previous section, however, work towards automation to directly support writers in their storytelling and story creation efforts for video games is relatively scarce. Nevertheless, progress is being made, and related work has many lessons to teach us, even if direct support for writers is not being offered.

One notable work is ScriptEase [6], an innovative pattern and template-driven approach, primarily aimed at in-game storytelling and behaviour control of non-player characters. In theory, the same framework could be extended to support cinematic and cut-scene generation, but this has not been done to date.

Work towards the <e-Game> engine [17] is also very promising. While primarily targeted at the development of adventure games, the XML-based <e-Game> language could be used to assist in the creation of cinematics and cut-scenes. The language, however, is geared towards game creation, and was not specifically designed with cinematic and cut-scene creation in mind. (In fact, it would appear from [17] that cinematics and cut-scenes are intended to be handled using pre-rendered movies instead of being specified and acted out by the engine itself.)

Interesting work also comes in the form of Bubble Dialogue [5], 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 game writers.

The Behavior Expression Animation Toolkit (BEAT) [3] is also relevant to story automation for games. Text is input to the system to be spoken by an animated character. As output, speech is generated, along with synchronized nonverbal behaviours that appropriately match the text according to rules based on human conversational patterns. This system is quite powerful and flexible, but as noted in [3], lacks many of the elements necessary to provide a complete performance on its own. It is designed, however, to plug into other systems for this purpose, and so could rely upon our own Reusable Scripting Engine for this support. Likewise, our system could benefit by having interesting and appropriate behavioral animations that are made possible by BEAT, and not available in our current prototype. The work is quite complementary.

Work towards interactive storytelling in games, such as the work in [8,11,15] and other examples discussed in [14], is also related, in that it involves story automation and story representation. In this case, automation tends to involve the synthesis of story emerging from the interactions between player and non-player characters in the game, with artificial intelligence controlling the non-player characters, according to authored constraints on behaviour. Our work, on the other hand, does not deal with interactivity, and so storytelling is driven entirely by the originally authored story. As a result, story representation for interactive stories can be significantly more complex, as additional elements are required to support and specify interactivity. This makes the process of story creation for interactive stories more like programming and, consequently, less friendly to writers with little or no programming experience. Our approach, on the other hand, avoids this complexity and burden on writers for cut-scenes and cinematics, where interactivity in the story is not required.

Other related work can be found in an interesting commercial video game entitled The Movies [13]. 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 writers to use in crafting stories for use in other games.

From an industrial perspective, as noted in [6,17], the video games industry has adopted a variety of standard and custom languages to be used in the development of games. These languages are used for many purposes, including the scripting of cinematics and cut-scenes. Unfortunately, while these languages improve and simplify matters somewhat, they are still rather complex and technical in nature. Consequently, writers still must rely upon at least some programming talent to integrate their stories into games [6].

In the end, much work is still required to assist writers in the story creation process for video games.

3. STORY SCRIPTING FOR VIDEO GAMES

Stories must be properly scripted to be automatically presented within a video game. This scripting identifies characters, dialogue, stage directions, setting, and other elements common to

traditional dramatic pieces. Fortunately, writers must already define these elements for cinematics and cut-scenes constructed according to traditional story creation processes [2], so the need for this information is not a new imposition created by automation.

For automation to be effective, however, stories must be scripted in a precise and formal manner to avoid potential ambiguity and confusion over the interpretation of the script by the software automating its presentation. Consequently, there is a need to provide a structured and standardized approach to scripting for storytelling within video games for automation efforts to be successful.

3.1. Encoding of Dramatic Pieces

As mentioned above, story scripts for games contain largely the same elements that can be commonly found in traditional dramatic pieces. Consequently, instead of developing our own custom language for specifying stories for games, as is frequently done in the literature in this area, we turn to efforts towards the standardization of encoding dramatic pieces, led by the Text Encoding Initiative (TEI).

These efforts have led to the development of an XML-based specification for marking up various kinds of texts, including dramatic pieces [18]. 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.

XML, however, is not the most natural or convenient method of expression for writers to use in creating their stories. Requiring writers to produce stories with manually embedded TEI tags needlessly complicates the process, and could be a barrier to story creation. 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 [20]. Several of these packages plug into existing word processing or office productivity software, or otherwise work with this software, to ensure that writers can work with familiar tools and still 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 slightly tailored guidelines for automation purposes, with some modifications and extensions to the existing TEI guidelines. This was necessary for several reasons:

- TEI guidelines require information that does not quite make sense or fit well within the realm of cinematics or cut-scenes for video games.

- Many elements in the TEI guidelines are not formal or precise enough for our purposes. For example, stage directions in the TEI guidelines are too open and too flexible; additional structure and formality needed to be applied to ensure these directions could be followed automatically.

- Additional elements were required to link game content and assets into story scripts. For example, character models, scenery, and other art assets must be identified and linked with corresponding elements within scripts, as well as audio assets for music, voice-overs, and so on.

- Other new elements were necessary to make managing story content within a game easier. For example, dialogue elements needed to be in manageable units that could be displayed on-screen and linked with voice-overs as necessary.

3.2. Story Scripting Elements

The final collection of story scripting elements used in our current work was derived from two sources. The first source, naturally, was the TEI guidelines themselves. Since these guidelines were developed based on the collaboration of experts in dramatic writing, they provide a reasonable set of elements with which to begin. The second source was an examination of cinematics and cut-scenes from several commercial video games of various genres from multiple platforms. This second source validated the elements from the first source, and identified refinements and other elements necessary, as discussed in the previous section.

The primary story scripting elements defined in our current work are depicted in Figure 1, showing the overall structure to our scripting schema. The main elements of this schema are briefly discussed in the subsections that follow. A full XML specification of this schema and a complete discussion of all of the story scripting elements defined within the schema can be found in [21].

3.2.1. The Performance

The Performance element is the top-level element for a cinematic or cut-scene. It identifies the name of the performance, as well as its place in the game. Its main purpose is to contain the Front Matter and Body for the cinematic or cut-scene, which provide the actual specification for the performance.

3.2.2. Front Matter

Front Matter is used to contain definitions of all of the elements to be used when the performance is acted out according to specifications in the Body, as described below. This includes the cast list for the performance, as well as a list of set materials, overlays, dialogue elements, sound effects, music, and voice-overs. This also includes a media element specifying, among other things, where media files for the various elements can be found.

The cast list defines every character that makes an appearance during the performance, as well as narrator characters that are heard but not necessarily seen. Each character may have multiple models, allowing them to appear in multiple costumes or multiple poses during the performance. The set list defines backdrops for the scenes of the performance. Overlays are essentially set elements that can be placed within a scene to create depth as characters can move both in front of and behind them. Dialogue

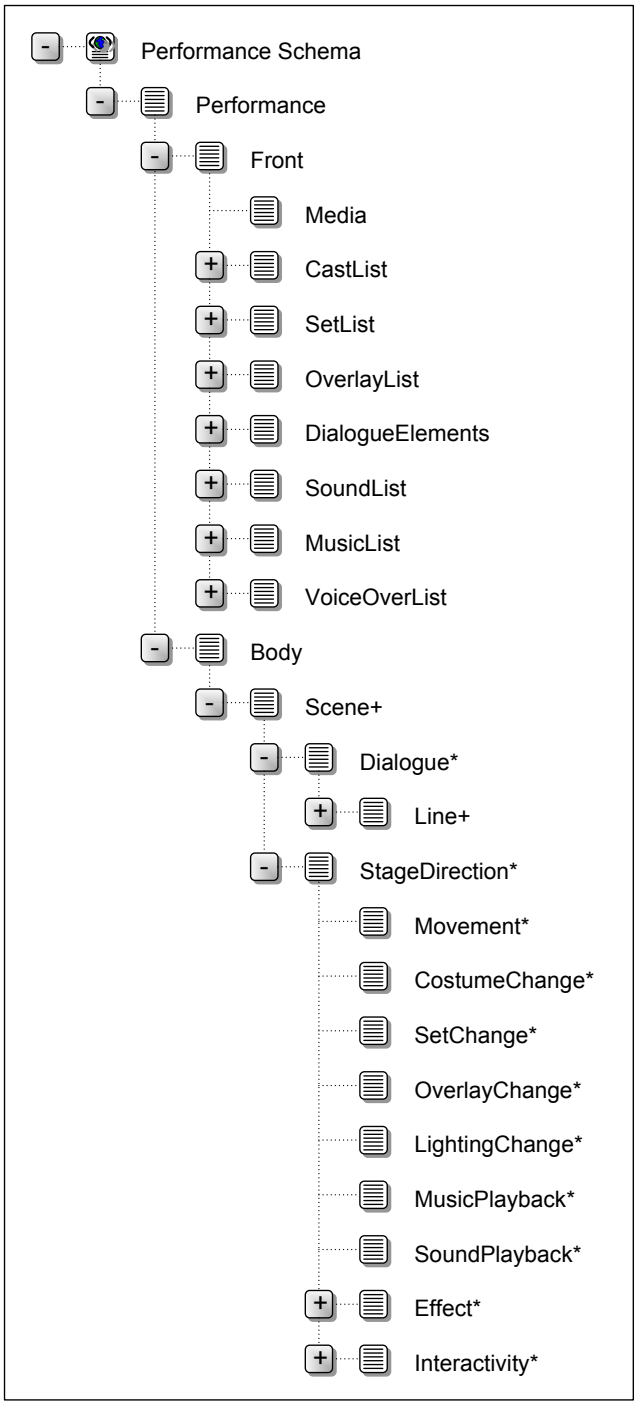

Figure 1. Schema Structure for Specifying Story Elements for Cinematics and Cut-Scenes.

Elements consist of dialogue areas into which text is rendered, definitions of fonts used to render text, and various options defining how and where this is done on-screen. The sound effect list identifies sounds that can be queued at any point throughout a performance. The music list defines background music that can be played or looped during a performance. Lastly, the voice-over list specifies speech effects that can be linked to lines of dialogue that appear in the Body of the performance.

3.2.3. The Body

The Body of a performance is used to specify the sequence of actual activities that constitute the performance using the elements defined in the Front Matter. The Body essentially consists of one or more scenes, based on the sets defined in the Front Matter. Scenes, in turn are composed of collections of dialogue elements and stage directions.

Dialogue elements are used to specify the narration and conversational elements in the performance. Each dialogue element is tied to a particular speaker (either an on-screen character or narrator), and consists of lines. Lines contain text, references to voice-overs to contain the narration or conversation of the dialogue, or both. Each dialogue element also carries with it a tone, allowing dialogue to be linked to changes in characters or dialogue areas to indicate the emotional context of the dialogue.

Stage directions are used to define all of the non-dialogue activities that occur during the performance of a cinematic or cut-scene. Movement directions bring characters on-screen, move them while on-screen, and move them off-screen when their part in the performance is done. Costume changes substitute the active character models in use for the characters in the performance. These can be used to make costume changes, as the name implies, or can be used for other purposes, such as changing the tone used by a character in dialogue. Set and overlay changes switch the current backdrop for the scene or alter the presence or positioning of items in the scene. Lighting changes adjust the lighting on scenery objects, characters, or both by increasing or decreasing light intensity, or by changing the colour of the lighting that is applied. Music and sound playback elements are used to control the playback of music and sound effects respectively.

There are also a variety of effects elements that are capable of providing a variety of miscellaneous effects during a performance. These include atmospheric effects (such as fog or smoke), transformation effects (such as flipping or rotating objects), and various special effects (like object transparency).

Lastly, interactivity elements are used to enable various types of interactions with the player of the game as they watch the cinematic or cut-scene in question. This includes pausing the performance for a period of time, waiting for user input (such as a key press or mouse button to indicate that the performance should proceed), and allowing the user to cancel the performance before it has ended.

4. PROOF OF CONCEPT

Having defined the story scripting elements of our current work in the previous section, in this section we examine the design and implementation of a proof of concept software engine capable of processing these elements and presenting a cinematic or cut-scene automatically as a result. This software engine is called the Reusable Scripting Engine.

4.1. Engine Design

To illustrate our design, we present the architecture of our engine in Figure 2, and then proceed to discuss the major modules of this architecture in the subsections below.

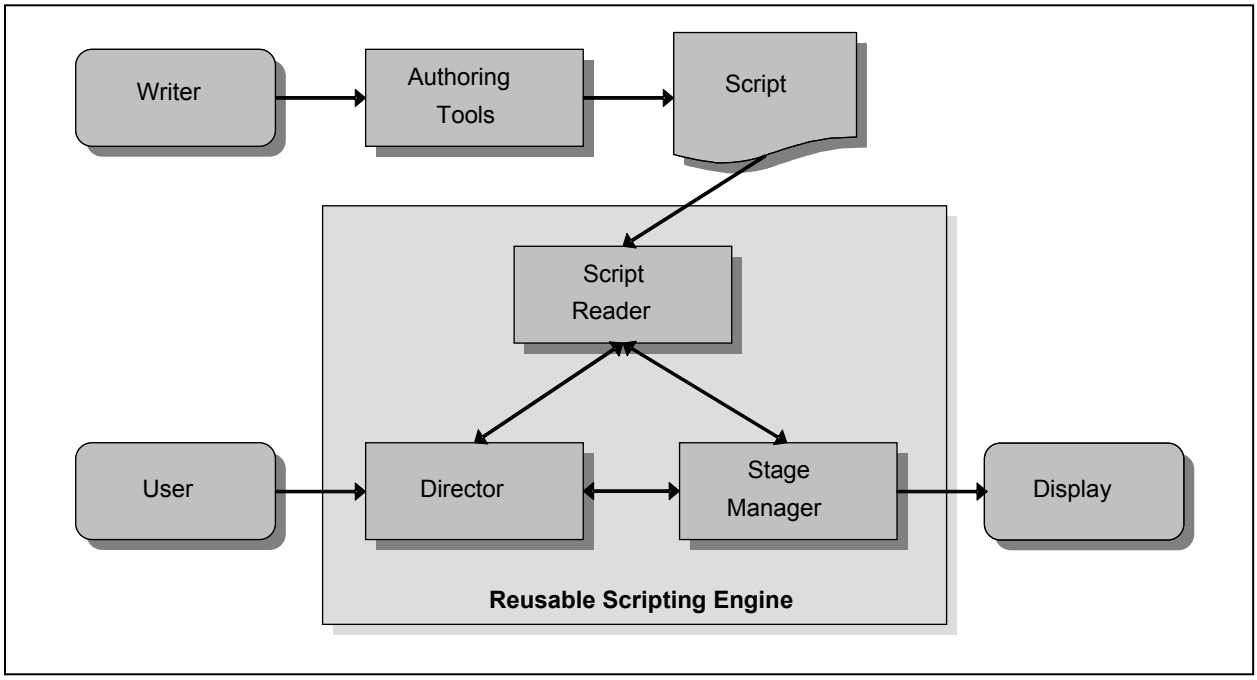

Figure 2. Reusable Scripting Engine Architecture.

4.1.1. 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 interactivity control to regulate the flow of the cinematic or cut-scene, as well as any other access required to the engine.

4.1.2. Script Reader

As the name implies, the Script Reader module reads in the story script, crafted by a writer using an appropriate authoring tool, 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 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 actions from the script, creating a performance, and passes this performance on to the Director module to have the performance executed.

4.1.3. 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 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.

4.2. Engine Implementation

Based on the architecture discussed in the previous section, we have implemented a prototype engine for Microsoft Windows XP, written in C# using Microsoft Visual Studio 2005 Professional Edition. To enable script processing, Microsoft's XML Software Development Kit was used, as it provides easy to use and robust XML processing and handling facilities when working in this environment.

For graphics and audio support, Microsoft DirectX was used. This provided us with clean, standard, and efficient support for both 2D and 3D graphics, as well as audio support, all in a single package. This also resolved outstanding issues from our earlier prototype [16] that primarily arose from our choice of rendering engine at the time. Consequently, this engine prototype is significantly more robust, efficient, and feature-rich than its predecessor.

Our engine implementation provides both a standalone processor that can generate cinematics and cut-scenes on its own, and a module that can be linked in with other code. These options provide developers with flexibility in how they integrate the engine into an existing game project.

Our implementation choices are also compatible with Microsoft's XNA Game Studio Express, meaning that we can target both the Windows platform and the Xbox 360 with our engine. While we have primarily carried out development on the Windows platform thus far, Xbox 360 support is currently under investigation as well.

```
<scene id="standardProcedure"
   setID="consultationRoom"
   initialLightingLevel="0">
   <stageDirection>
      <musicPlayback id="bgMusic"
         loop="on"/>
      <lightingChange level="100"
         subject="scenery"
         duration="1"/>
      <pause duration="2"/>
   </stageDirection>
   <dialogue speaker="narrator">
      <line>- Hope Hospital,
         Consultation Room - </line>
   </dialogue>
   <stageDirection>
      <waitFor event="anyInput"/>
      <soundPlayback id="beep" />
      <movement castID="mary"
         type="enterHorizontal"
         startLocation="offRight"
         endLocation="onRight"
         speed="1000" />
   </stageDirection>
   <dialogue speaker="mary">
      <line>The patient has been
         moved to \nthe pre-op
         area.</line>
   </dialogue>
   <stageDirection>
      <waitFor event="anyInput"/>
      <soundPlayback id="beep" />
      <lightingChange level="25"
         subject="scenery"
         duration="1"/>
      <pause duration="1"/>
   </stageDirection>
   <dialogue speaker="narrator">
      <line>Mary Fulton, age 39: Hope
         Hospital's \nveteran
         surgical assistant.</line>
   </dialogue>
   <stageDirection>
      <waitFor event="anyInput"/>
      <soundPlayback id="beep" />
   </stageDirection>
   <dialogue speaker="narrator">
        <line>She's kind and well-
           liked, so nobody\n
           mentions she tends to
           ramble too much.</line>
   </dialogue>
```

Figure 3. Scripting Used to Replicate Standard Procedure Scene from Trauma Center: Second Opinion.

5. EXPERIENCES TO DATE

Initial experimentation with our Reusable Scripting Engine involved recreating scenes from movies and television shows such as the Princess Bride [10] and The Simpsons [19], as discussed in [16]. While this experimentation demonstrated that our engine could create cinematics and cut-scenes, it did not necessarily validate its suitability for use in video games.

To do so, we selected a game from the set of games with cinematics and cut-scenes initially examined as discussed in Section 3.2, and replicated some of its scenes. The game selected for our current work was Trauma Center: Second Opinion [1], developed by Atlus for the Nintendo Wii platform. This game was chosen as it would require a robust and rich set of story scripting elements to be adequately replicated, and consequently provided a suitable test of our engine's capabilities.

Figure 3 presents an excerpt from the scripting that was used in replicating the Standard Procedure cut-scene that appears before the first operation in the game. Figure 4 contains screenshots illustrating how this script would appear when performed. This portion of the cut-scene is performed as follows:

1. The scene begins by preparing the set for the cut-scene, the consultation room. Initially, lighting is set to a level of 0%, indicating that the set will be dark to begin with. Stage directions then begin playback of background music, set to loop indefinitely. A lighting change occurs, to raise set lighting to a level of 100%, to fully illuminate the set. This is done over a period of 1 second. This is followed by a pause of 2 seconds before the performance continues. This results in the scene as depicted in Figure 4 (a).

2. Dialogue then begins, with the narrator introducing the scene, resulting in what appears in Figure 4 (b), with dialogue text appearing in a shaded area at the bottom of the screen.

3. Stage directions have the performance pause to wait for input from the player, to ensure they have had the chance to read the dialogue. Any input is acceptable to continue the scene. A beep sound effect is then played to acknowledge the input, as was done in the original game. Mary is then directed to quickly enter from stage right and stay on the right half of the scene, resulting in what is depicted in Figure 4 (c).

4. Mary then says her line in her default tone, since no tone was specified. Since no voice-overs occurred in the original game, none were included with this line of dialogue either. This results in what appears in Figure 4 (d).

5. At this point, the scene pauses as discussed in Step 3, and a lighting change occurs to dim scenery lighting to 25%. The lighting on Mary, however, is preserved, causing her to stand out while the narrator introduces her in the next line of dialogue, as shown in Figure 4 (e).

6. The scene pauses once again as discussed above, and the narrator completes the introduction of Mary, as depicted in Figure 4 (f). After this, lighting is restored to normal levels, and the scene continues appropriately.

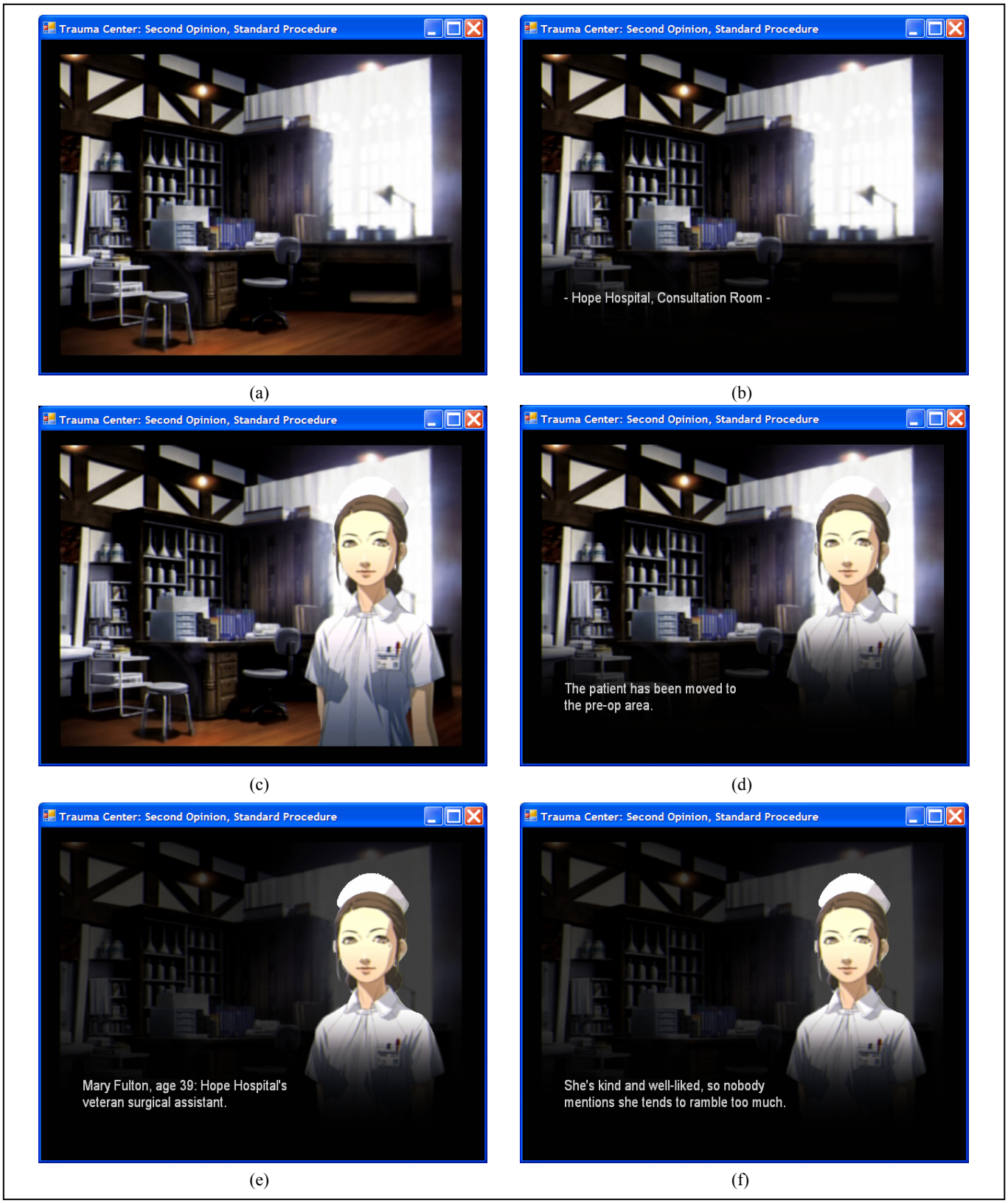

Figure 4. Screenshots from Replication of Standard Procedure Scene from Trauma Center: Second Opinion.

In the end, our Reusable Scripting Engine allowed us to faithfully, accurately, and easily replicate scenes from the original game. The story script was authored using the XML editing facilities in Visual Studio 2005 Professional Edition. Even though this package has meager XML editing capabilities in comparison to other packages, it took very little time and effort to construct the scripts.

All in all, our experiences with our Reusable Scripting Engine prototype have been very positive, and it demonstrates tremendous possibilities for its use in the future.

6. CONCLUSIONS AND FUTURE WORK

Storytelling is an important aspect of modern video games, and plays a central role both in drawing in players initially and in keeping them playing over the long term. With the success or failure of games depending on their story elements, it is becoming increasingly important to provide tools and supports to allow writers to directly produce story content for games, without requiring programming background and expertise. This allows stories for 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 current work in this area addresses this need for tools and supports by providing story scripting elements and a software engine capable of using these elements to automatically produce cinematics and cut-scenes for video games. By feeding authored story content directly into our Reusable Scripting Engine, writers no longer need to depend upon programming talent to have their stories acted out, which can have significant benefits. Results from using our prototype engine to date have been quite positive and show great potential for the future.

Possible directions for continued work in this area include the following. Replicating cinematics and cut-scenes from other video games is an important next step. This will not only provide further validation of our approach and engine, but it will also help to uncover additional stage directions, effects, or other elements that should be supported in our work. Support for animated characters, set elements, and effects is also important. Static characters with little or no animation are still in use in many games today (such as Trauma Center: Second Opinion), 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 through our use of DirectX. This will require the addition or refinement of stage directions to enable our scripting to work in a truly 3D space. There is currently considerable interest in dynamic story elements in video games that allow the flow of story to change depending on in-game events. Our engine can and should be extended to support these efforts. Our Reusable Scripting Engine should be ported through XNA to the Xbox 360. This platform is attractive to academic, independent, and hobbyist developers, and so providing automated storytelling support would be very beneficial to development efforts in this area.

7. REFERENCES

[1] Atlus. *Trauma Center: Second Opinion*. Published by Atlus, 2006.

[2] Bateman, C. *Game Writing: Narrative Skills for Videogames.* Charles River Media. 2007.

[3] Cassell, J., Vilhjalmsson, H. and Bickmore, T. BEAT: The Behavior Expression Animation Toolkit. *SIGGRAPH 2001 Conference.* Los Angeles, California, August 2001.

[4] Chandler, R. *Game Writing Handbook*. Charles River Media. 2007.

[5] Cunningham, D., McMahon, H. and O'Neill, B. "Bubble Dialogue: A New Tool for Instruction and Assessment", *Educational Technology Research and Development, Volume 40, Number 2*. 1992.

[6] Cutumisu, M., Onuczko, C., McNaughton, M., Roy, T., Schaeffer, J., Schumacher, A., Siegel, J., Szafron, D., Waugh, K., Carbonaro, M., Duff, H. and Gillis, S. "ScriptEase: A Generative/Adaptive Programming Paradigm for Game Scripting". *Science of Computer Programming, Volume: 67, Issue: 1*. June, 2007.

[7] Davies, M.. *Designing Character-Based Console Games*. Charles River Media. 2007.

[8] El-Nasr, M. Interaction, Narrative, and Drama Creating an Adaptive Interactive Narrative using Performance Arts Theories. *Interaction Studies, Volume 8, Number 2*, 2007.

[9] Glassner, A. *Interactive Storytelling: Techniques for 21st Century Fiction*. A K Peters Limited. 2004.

[10] Goldman, W. *The Princess Bride*. 20th Century Fox. September 1987.

[11] Gordon, A., van Lent, M., van Velsen, M., Carpenter, M. and Jhala, A. Branching Storylines in Virtual Reality Environments for Leadership Development. \ *Sixteenth Innovative Applications of Artificial Intelligence Conference (IAAI-04)*, San Jose, California, July 2004.

[12] Krawczyk, M. and Novak, J. *Game Development Essentials: Game Story and Character Development*. Thomson Delmar Learning. 2006.

[13] Lionhead Studios. *The Movies*. Activision. 2005.

[14] Magerko, B. A Comparative Analysis of Story Representations for Interactive Narrative Systems. *Third Annual Artificial Intelligence for Interactive Digital Entertainment Conference.* Marina del Rey, California. 2007.

[15] Mateas, M. and Stern, A. Facade: An Experiment in Building a Fully-Realized Interactive Drama. *Game Developer's Conference*, San Francisco, California, March 2003.

[16] McLaughlin, M. and Katchabaw, M. "A Reusable Scripting Engine for Automating Cinematics and Cut-Scenes in Video Games". *Loading ... The Journal of the Canadian Game Studies Association, Vol. 1, No. 1*, May 2007.

[17] Moreno-Gera, P., Sierra, J., Martínez-Ortizb, I. and Fernández-Manjóna, B. "A Documental Approach to Adventure Game Development". *Science of Computer Programming, Vol. 67, Issue 1*. June, 2007.

[18] Sperberg-McQueen, C. and Burnard, C., (eds). *Guidelines for Text Encoding and Interchange: XML-compatible Edition*. Published for the TEI Consortium by the Humanities Computing Unit, University of Oxford. 2004.

[19] Stern, D. "Duffless." *The Simpsons Episode 9F14.* 20th Century Fox Broadcasting Company. February1993.

[20] The TEI Consortium. "TEI Software". *Available at: http://www.tei-c.org/Software*. Last accessed July 2007.

[21] Zhang, W. "Reusable Storytelling Engine". *MSc. Directed Study Report, The Department of Computer Science, The University of Western Ontario*. June 2007.