AUTOMATING CINEMATICS AND CUT-SCENES IN VIDEO GAMES THROUGH SCRIPTING WITH ACTIVE PERFORMANCE OBJECTS

V. Bonduro and M. Katchabaw
Department of Computer Science
The University of Western Ontario
London, Ontario, Canada N6A 5B7
vbonduro@uwo.ca, katchab@csd.uwo.ca

KEYWORDS

Storytelling, automation, active performance objects, story scripting, cut-scenes, cinematics, video games

ABSTRACT

Storytelling is widely recognized as an important element of modern video games. Unfortunately, it can be exceedingly difficult for writers to directly author and integrate story content into games on their own. Instead, they must rely on programmers, artists, and other personnel on the development team to implement their stories. This complicates the story creation process needlessly, increases costs, reduces time available for other tasks, and causes writers to lose creative control over their works. As a result, tools and supports are necessary to enable writers to generate story content for games directly, without the need for outside assistance.

This paper continues our earlier work that used 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. Our current work goes beyond this earlier work to enable more flexible, dynamic, and enriched performances through the use of Active Performance Objects. This paper presents these advancements, as well as our most recent experiences with using this engine to recreate cinematics and cut-scenes from a variety of existing commercial video games.

INTRODUCTION

Storytelling can be one of the most important and compelling aspects of modern video games (Bateman 2007; Glassner 2004; Krawczyk and Novak 2006), and in some cases is regarded as one of their most defining aspects (Davies 2007). 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, the importance of storytelling in games will only increase (Chandler 2007).

While story creation is naturally the responsibility of writers on the development team (Bateman 2007; Moreno-Gera et al. 2007), these writers traditionally must work with programmers, artists, and others on the team to integrate story content into the game being developed due to the complexity involved and domain expertise required. This results in the traditional story creation process depicted in Figure 1.

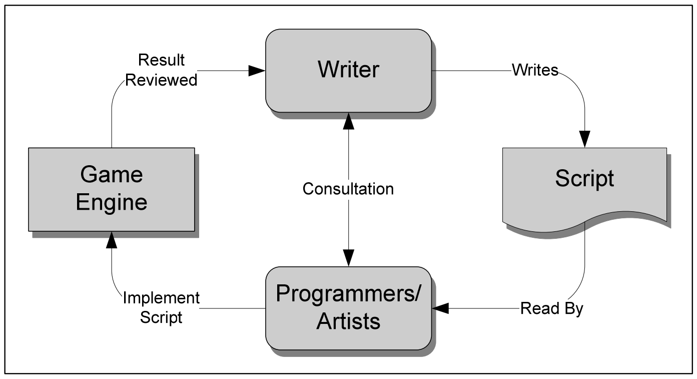


Figure 1: Traditional Story Creation Process

This process, however, can be expensive in terms of budget and scheduling resources for the programmers and artists involved (Cutumisu et al. 2007), which is problematic considering the limitations often in place in creating story content for games (Bateman 2007). Furthermore, this introduces a gap between storyteller and story implementation (Cutumisu et al. 2007), in which mistakes, miscommunication, and differences in opinions and vision can impact the creative process and overall story quality as a result.

For these reasons, a simpler, more streamlined story creation process for games is necessary—a need recognized for some time by industry practitioners (Bateman 2007; Cutumisu et al. 2007). Automating storytelling can alleviate these issues by allowing writers to tell their stories in games with minimal outside 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, as shown in Figure 2.

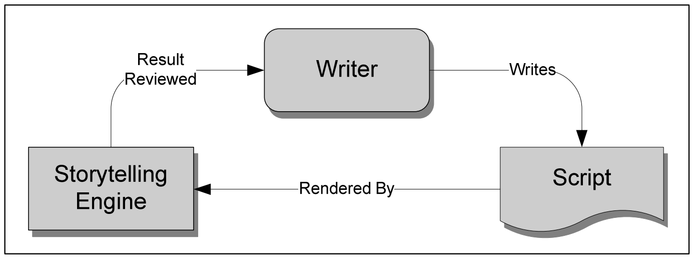


Figure 2: An Automated Approach to Story Creation

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 and Novak 2006). 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 (McLaughlin and Katchabaw 2007; Zhang et al. 2007).

Our earlier work primarily focused on the core elements of the Reusable Scripting Engine, and scripting elements for representing story. It was found in practice, however, that this earlier work was not flexible or powerful enough to support much of what is found in cinematics and cut-scenes in video games. Such performances are more varied and dynamic, with rich and active content, and so serious changes to our engine were required to achieve higher levels of quality in our work.

This paper introduces and discusses the details of our new approach to automating storytelling, using Active Performance Objects to address the issues discussed above, as well as the refactoring of our Reusable Scripting Engine platform to support this new approach. This paper also describes our most recent experiences through using our new engine to recreate cinematics and cut-scenes from a variety of commercial games.

The remainder of this paper is organized as follows. We begin by presenting related work in this area, both from research and industrial perspectives. We then describe the design and engine architecture of the approach to automated storytelling taken in our own work. We then present the implementation of our proof of concept system, the Reusable Scripting Engine, and discuss our experiences from using this in recreating cinematics and cut-scenes from a variety of commercial video games. Finally, we conclude this paper with a summary and a discussion of directions for future work.

RELATED WORK

This section discusses relevant related work, both from research and industrial perspectives. Unfortunately, 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 (Cutumisu et al. 2007), 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 (Moreno-Gera et al. 2007) 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, according to (Moreno-Gera et al. 2007), it would appear that cinematics and cut-scenes are intended to be handled using pre-rendered movies instead of being acted out by the engine itself.)

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 game writers.

The Behavior Expression Animation Toolkit (BEAT) (Cassell et al. 2001) 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 (Cassell et al. 2001), 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 (El-Nasr 2007; Gordon et al. 2004; Mateas and Stern 2003) and other examples discussed in (Magerko 2007), 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 (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 writers to use in crafting stories for use in other games.

From an industrial perspective, as noted in (Cutumisu et al. 2007; Moreno-Gera et al. 2007), the video games industry has

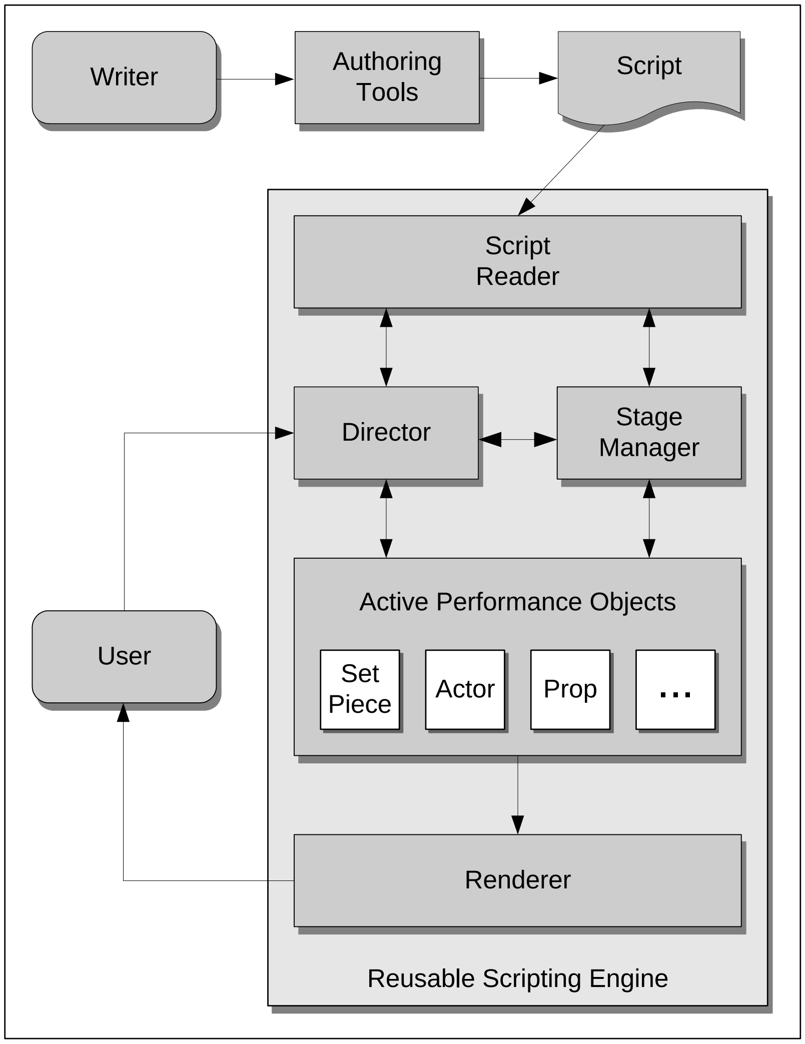


Figure 3: Reusable Scripting Engine Architecture

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 (Cutumisu et al. 2007).

While the literature in this area has made many interesting and important contributions to storytelling in video games, much work is still required to fully assist writers in the story creation process.

DESIGN AND ENGINE ARCHITECTURE

As shown earlier in Figure 2, our approach to the automation of storytelling in video games is driven largely by a story script which is written by a writer and then rendered and acted out using a software engine, the Reusable Scripting Engine in our case. The architecture of this engine and the flow of story content through it are depicted in Figure 3, and discussed in the sections that follow.

Writer

The writer is the creator of story content for a game, and as such is primarily responsible for the creation of a script that captures this story, defining the setting and characters involved in the story and complete with all of the dialogue and stage directions required to enact the story. Fortunately, as shown in Figure 1, writers must already script such story elements for cinematics and cut-scenes constructed according to traditional story creation processes (Bateman 2007), so the need for this information is not a new imposition created by the automation.

Script

For automation to be effective, 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 approach to scripting for storytelling within video games for automation efforts to be successful.

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 standardization, led by the Text Encoding Initiative (TEI). These efforts have developed an XML-based specification for marking up various kinds of texts, including performances and dramatic pieces (The TEI Consortium 2008a). 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.

As discussed in (McLaughlin and Katchabaw 2007), however, some extensions and modifications were needed to the base TEI guidelines to adapt them for use in video game scripts, to provide more formality and precision where it was needed, to link game content and assets into story scripts, and to filter out elements that were unnecessary in this context. A complete discussion of the various scripting elements supported by our Reusable Scripting Engine can be found in (Zhang et al. 2007).

As an example, consider the story script excerpt in Figure 4. This script is for a scene from the video game Trauma Center: Second Opinion for the Nintendo Wii platform (Atlus 2006), and is used in experimentation presented later in this paper. This story script is interpreted as follows:

1. The scene begins by preparing the set for the 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.

2. Dialogue then begins, with the narrator introducing the scene.

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. The actor Mary is then directed to quickly enter from stage right and stay on the right half of the scene.

4. Mary then says her line in her default tone, since no tone was specified. (The results of this can be found later in Figure 5.) Since no voice-overs occurred in the original game, none were included with this line of dialogue either.

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.

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

```
<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 4: Scripting Used to Recreate Standard Procedure Scene from Trauma Center: Second Opinion

Authoring Tools

As one can imagine, XML is not the most natural or convenient method of expression for writers to use in authoring their stories. Requiring writers to produce stories with manually embedded TEI tags needlessly complicates the process, and imposes a barrier to story creation. To assist in the process of working with TEI tags, there are numerous authoring tools available that adhere to TEI guidelines for importing existing works or writing them from scratch (The TEI Consortium 2008b). Several of these packages plug into existing word processing 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.

Script Reader

As the name implies, the Script Reader module in the Reusable Scripting Engine 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 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 lists of all of the set pieces, actors, and props involved in the performance, along with a stream of actions from the script that carries out this performance. These actions include dialogue, stage directions, and guidelines for managing interactivity with the user. This information is then passed on to the Director module to have the performance executed.

Director

The primary role of the Director in the engine is to control the flow of a performance. In doing so, the Director manages 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, as discussed below, could either be the player of the game in question or the game itself, depending on the context. 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.

Stage Manager

The Stage Manager module is responsible for ensuring that the performance is carried out according to the directions of the Director, including what to do, how to do it, and when to do it. The Stage Manager also reports back to the Director on the status of the production as it progresses.

In our earlier work in (McLaughlin and Katchabaw 2007; Zhang et al. 2007), the Stage Manager was directly responsible for the coordination and rendering of all of the various elements of the performance on its own. While this approach was simple and straightforward, it also lacked the flexibility and expressive power to deliver rich performances with a variety of active and dynamic content, such as animations.

To resolve this problem, the Stage Manager was redesigned so that it was no longer directly responsible for the rendering of the performance. Instead, these responsibilities were delegated to a collection of Active Performance Objects and a dedicated Renderer module, with the Stage Manager responsible for managing the Active Performance Objects according to the directions of the Director.

Active Performance Objects

In our earlier work, set pieces, props, and actors only existed as data contained within the Stage Manager. As necessary, the Stage Manager consulted this data to carry out the performance.

In our current work, each set piece, prop, and actor is encapsulated by an Active Performance Object. Each such object is now responsible for its own use and behaviour in the context of the performance according to guidance from the Stage Manager. Furthermore, each Active Performance Object is responsible for managing and maintaining its own data, current state, and associated assets, to ensure that it is ready for rendering by the Renderer when the time to do so comes.

If an Active Performance Object is dynamic and changes over time, it contains its own thread of execution to assist in the above tasks as necessary. Coordination between Active Performance Objects is handled by the Director or Stage Manager, depending on the coordination required.

Through proper use of Active Performance Objects, a performance can now contain a large collection of independent or cooperating active elements that are all at work simultaneously. This provides a considerable amount of power and flexibility in constructing a rich and high quality performance. For example, it is now possible through the use of Active Performance Objects to have an animated set, with multiple actors moving around in the background, while actors in the foreground engaged in dialogue, complete with voiceovers synced with facial animations. This type of rich performance was simply not possible under our earlier engine.

Renderer

The Renderer module in the Reusable Scripting Engine is ultimately responsible for the rendering of the performance to the user. It does so by iterating through and working with the collection of Active Performance Objects and composing a scene from these objects based on their current states in the performance.

To do its work, the Renderer also has its own thread of execution. This allows it to work independently of the Active Performance Objects to collect and push graphics and audio data out to system devices when this data is required.

User

As mentioned earlier, the user of the Reusable Scripting Engine can either be the player of the game or the game itself, or perhaps both at the same time. This, naturally, depends on the context and the game in question.

The player of the game can interact with the Director module in the engine to pause or skip the performance, tune performance options, and so on. The player also ultimately watches the performance as it is rendered by the Renderer module. The game itself is a user of the engine in that the game may also need to control the flow of the performance, depending on the situation. Furthermore, the game may also need to tune performance options at various points during its life time.

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, with .Net Framework 2.0. The prototype has also been tested and runs perfectly on the various versions of Microsoft Windows Vista.

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.

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.

EXPERIENCES TO DATE

Initial experimentation with our Reusable Scripting Engine in (McLaughlin and Katchabaw 2007) involved recreating scenes from movies and television shows such as the Princess Bride (Goldman 1987) and The Simpsons (Stem 1993). To demonstrate the engine's suitability for use in video games, our work in (Zhang et al. 2007) successfully applied our engine to the game Trauma Center: Second Opinion, mentioned earlier in this paper.

To demonstrate and evaluate the capabilities of our new engine architectures with Active Performance Objects, we recreated cinematics and cut-scenes from a variety of different commercial games, from various genres and platforms, using a variety of artistic and presentation styles. In doing so, we were able to provide a suitable test of our engine's flexibility, expressive power, and functionality. Our experiences with three of these games are discussed in the sections below in detail.

Trauma Center: Second Opinion

In our first experimentation with the new version of the Reusable Scripting Engine, we started with the Trauma Center: Second Opinion performance used in our earlier work, as described above. This was done to ensure that the redesign of our approach to use Active Performance Objects was successful and did not impact the ability of the engine to carry out performances. As expected, no problems whatsoever were encountered in doing so.

While this initial experimentation with Active Performance Objects was successful, there was nothing in the performance that was active that required their enhanced capabilities. As a result, we extended and embellished our original Trauma Center: Second Opinion performance, to provide a more interesting test, as shown in the screen shot in Figure 5.

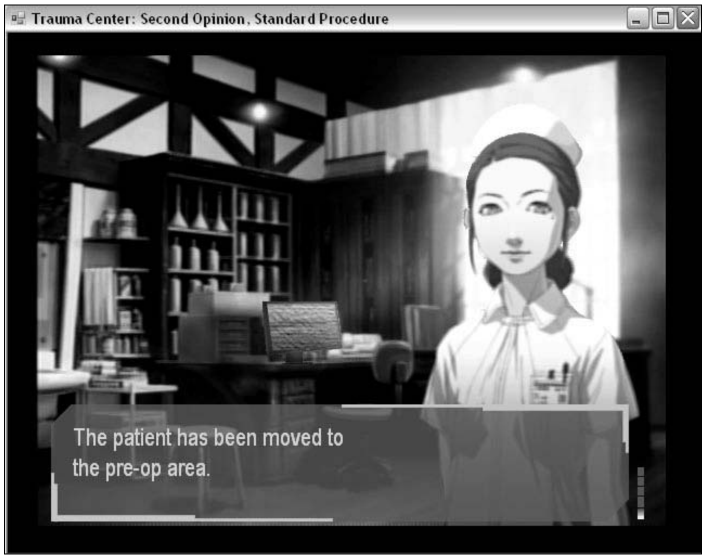

Trauma Center: Second Opinion, Standard Procedure
The patient has been moved to the pre-op area.

Figure 5: A Scene from a Performance from Trauma Center: Second Opinion Using the Reusable Scripting Engine

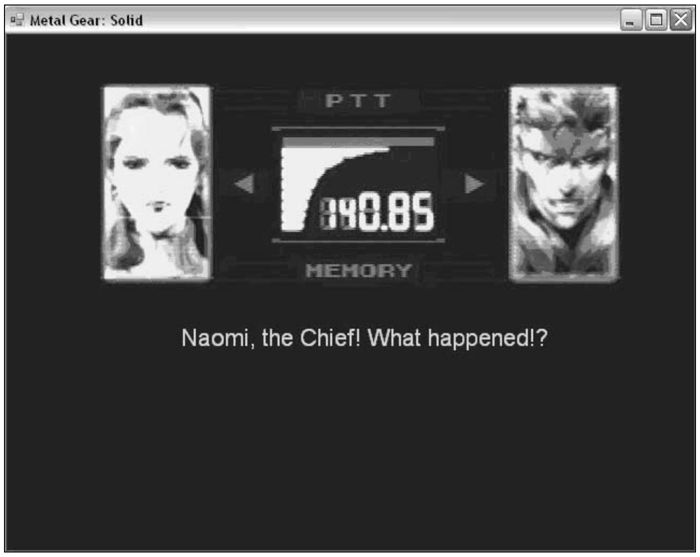
In the scene shown in Figure 5, we added a computer display as a prop that did not appear in the original performance, as seen in the middle of the figure. This display was animated with a changing image and a flicker effect that changed its illumination and that in the scene around it. These animations were controlled by the Active Performance Object that encapsulated the display prop.

Improvements were also made to the dialogue area visible at the bottom of Figure 5, improving its appearance, and adding an animated dialogue icon indicating that the user could advance through the performance. Additional dialogue rendering modes were added to allow the user to force the complete rendering of a line of dialogue before it was typed out character-by-character, as was done in the original performance.

All in all, the improved Reusable Scripting Engine handled these tests quite well in executing this performance.

Metal Gear Solid

While the Trauma Center: Second Opinion experiments were successful, they barely started to test the capabilities of the Active Performance Objects in the new engine. Consequently, we reconstructed a scene from Konami's Metal Gear Solid for the Sony PlayStation (Konami 1998).



Naomi, the Chief! What happened!?

Figure 6: A Scene from a Performance from Metal Gear Solid Using the Reusable Scripting Engine

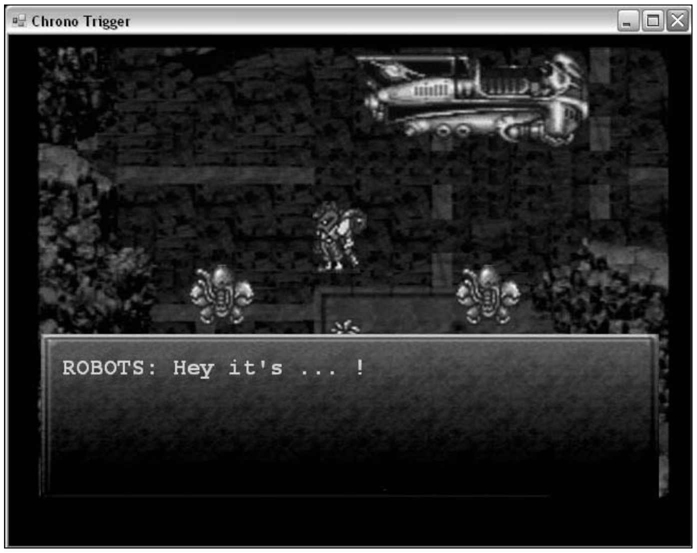
This scene is more complicated than the Trauma Center: Second Opinion performance, with an animated set, animated actors, voiceovers linked to dialogue, and so on, with each of these elements encapsulated by Active Performance Objects. As shown in the screen shot in Figure 6, the setting is the Codec communication system in the game, which has an animated signal indicator in the middle of the scene. The actors are both animated in several ways. First, their images expand at the beginning of the scene, as if they were in displays being turned on. Second, their images flicker and scroll with static lines throughout the scene, again to create the illusion as if they are on some sort of display screen. Finally, their faces are animated while delivering lines of dialogue, to make it look as if they are speaking. Each line of dialogue delivered is linked to a voiceover; this, together with the facial animation above, provides a reasonably impressive performance.

Constructing this scene also required the addition of new rendering and playback modes. Unlike Trauma Center: Second Opinion, whose cinematics and cut-scenes were driven by the user advancing the performance, the performance in Metal Gear Solid was intended to play out on its own, without interaction from the user. If the user interacted with the performance, however, it would switch to a user-driven mode. This also necessitated the development of new handlers to support a wider variety of interactions with the user.

In the end, the Reusable Scripting Engine was able to recreate the scene from Metal Gear Solid quite well, even though it is substantially different from the Trauma Center: Second Opinion scene. This demonstrates the flexibility and robustness of our approach.

Chrono Trigger

To further demonstrate the capabilities of the new Reusable Scripting Engine and its Active Performance Objects, we recreated a scene from Square Soft's Chrono Trigger for the Super Nintendo Entertainment System (Square Soft 1995). As can be seen from the screen shot in Figure 7, this game used a very different style and approach to story presentation in comparison to the other performances examined so far.



ROBOTS: Hey it's ... !

Figure 7: A Scene from a Performance from Chrono Trigger Using the Reusable Scripting Engine

A major difference in the Chrono Trigger scene is that there are now several animated actors involved in the scene, with all of them animated or moving at once, making the performance considerably more complex. The scene shown in Figure 7 contains eight such actors, although some are periodically obscured by the dialogue area. Each actor is again encapsulated by an Active Performance Object that manages its animation and movement, and coordinates its activities with the Director and Stage Manager in the engine, to ensure that the actors are moving and are animated in unison as necessary.

The range of movements required in the Chrono Trigger performance necessitated the development of new stage directions and new mechanisms for tracking and controlling movements in the engine. Previous scenes were relatively simple, with movement needs handled by simple directions such as "Enter, stage right" and "Exit, stage left". Chrono Trigger, on the other hand, required arbitrary actor movements, and so new methods were required to identify arbitrary movement targets in a scene and new stage directions were required to enable these movements to be scripted by the writer of the story.

Despite the additional complexities introduced by the Chrono Trigger story, the Reusable Scripting Engine was again able to faithfully recreate the scenes in its own performances quite well.

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 (Krawczyk and Novak 2006). 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 a Reusable Scripting Engine that is capable of producing high quality cinematics and cut-scenes for a wide variety of video games based on scripts provided by story writers. The use of Active Performance Objects in our current work enables the use of dynamic and active content in stories to create a richer experience for the user. Results from using our prototype engine to date have been quite positive, demonstrating the flexibility and expressive power of our approach to automating storytelling.

Possible directions for continued work in this area in the future include the following:

- Recreating cinematics and cut-scenes from other video games is still an important next step. This will not only provide further validation of our approach and engine, but it will also help to uncover further additions necessary to our work.

- Support for 3D cinematics and cut-scenes is also necessary, and is made 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. Fortunately, our recent experiences with the Reusable Scripting Engine, in particular in the construction of the Chrono Trigger performance, have given us insight into storytelling in an open 2D space that might carry over into a 3D space as well.

- 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.

REFERENCES

Atlus. 2006. *Trauma Center: Second Opinion*. Published by Atlus.

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

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

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

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

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. 2007. "ScriptEase: A Generative/Adaptive Programming Paradigm for Game Scripting". *Science of Computer Programming, Volume: 67, Issue: 1*. (June).

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

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

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

Goldman, W. 1987. *The Princess Bride*. 20th Century Fox. (September).

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

Konami Computer Entertainment Japan. 1998. *Metal Gear Solid*. Published by Konami.

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

Lionhead Studios. 2005. *The Movies*. Activision.

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

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

McLaughlin, M. and Katchabaw, M. 2007. "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).

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

Square Soft. 1995. Chrono Trigger. Published by Square Soft.

Stern, D. 1993. "Duffless." *The Simpsons Episode 9F14.* 20th Century Fox Broadcasting Company. (February).

The TEI Consortium. 2008a. "TEI P5: Guidelines for Electronic Text Encoding and Interchange." *Available at: http://www.tei-c.org/Guidelines/P5*. (Last accessed June).

The TEI Consortium. 2008b "TEI Tools". *Available at: http://www.tei-c.org/Tools*. (Last accessed June).

Zhang, W., McLaughlin, M., and Katchabaw, M. 2007. Story Scripting for Automating Cinematics and Cut-Scenes in Video Games. Proceedings of FuturePlay 2007. Toronto, Canada, (November).