Automating Content Analysis of Video Games

T. Bullen and M. Katchabaw
The University of Western Ontario
tbullen@uwo.ca, katchab@csd.uwo.ca

N. Dyer-Witheford
The University of Western Ontario
ncdyerwi@uwo.ca

Abstract

Content analysis of video games tends to be an extremely arduous task, involving the collection of a very large quantity of data and statistics detailing the experiences of gameplay. Nevertheless, it is an important process that supports many business, policy, social, and scholarly activities related to the games industry. Consequently, supports are clearly necessary to facilitate content analysis procedures for video games.

This paper discusses an innovative approach to automating content analysis for video games through the use of software instrumentation. By properly instrumenting video game software to enable data collection and processing, content analysis procedures can be either partially or fully automated, depending on the game in question. This paper discusses our overall approach to automation, as well as our experiences to date with Epic's Unreal Engine. Sample results from initial experiments conducted so far are also presented. These results have been quite positive, demonstrating great promise for continued work in this area.

Introduction

Content analysis of video games involve coding, enumerating, and statistically analyzing various elements and characteristics of games, including violence, criminal behaviour, offensive language, substance abuse, sexual activity, gender and racial inclusiveness, and so on. While content analysis has limitations, as demonstrated in (Hosti, 1969; Newman, 2004), it is invaluable in providing a quantitative assessment of games to complement more qualitative analysis, as recently suggested (Bogost, 2006). As such, content analysis is an important tool to scholars of game studies and other media issues; policy makers dealing with issues of regulation, ratings and censorship; psychologists dealing with media effects; developers and publishers producing games; and parents, educators and game players using these games.

When one applies traditional content analysis procedures, for example from television or film, to video games, several difficulties are encountered. These procedures are manual and tend to be time consuming and labour-intensive, with the potential to generate or accumulate a considerable volume of data. This results in analysis being hindered by limited play time, sometimes just the first level (Heintz-Knowles et al., 2001) or first few minutes (Brand &

Proceedings of CGSA 2006 Symposium

Knight, 2003), or, alternatively, playing very few games to have time for more thorough examinations (Grimes, 2003). Furthermore, traditional analyses often do not consider the effects of player interactivity and non-linearity in games, which can limit their accuracy unless these issues are explored more fully. These issues are further compounded by the speed with which games are released, and the rapid rate at which the medium evolves. It becomes quite difficult to conduct thorough analysis of a reasonable portion of games with the limited time and resources typically available for doing so. A solution to these problems is clearly needed.

This paper continues our earlier work in automating content analysis of video games (Bullen, Katchabaw & Dyer-Witheford, 2006), addressing the above problems by taking advantage of the fact that unlike other forms of media, video games are ultimately software executing on a computing device. Because of this, other software on the computing device can monitor game execution and collect and report much of the data required for content analysis, without the need for traditional manual methods. This provides a partial automation of the content analysis process, as a human subject is still required to play the games in question to generate the play experiences being monitored by the software. We can, however, go further and provide a full automation of the process in some cases by having software take the role of the player and generate gameplay experiences without human intervention. In providing these supports, automation effectively reduces the time, labour, and resources required to conduct a thorough content analysis. This allows longer and more representative analysis of more games, and allows analysis to be conducted more frequently. Automation also permits broader studies of interactive and non-linear play, with the potential for more data to be collected than through manual processes alone.

The approach taken in our work uses a framework of software instrumentation components to facilitate content analysis of games in a minimally invasive fashion. This instrumentation can be easily embedded within existing game software to enable the monitoring and control required to conduct a thorough analysis of a game. As a proof of concept, we have used our framework to instrument Epic's Unreal Engine (Epic Games, 2005), a popular engine used in the development of numerous games in professional, independent, hobbyist, and academic settings. Through instrumenting the engine, we are able to automate the content analysis of any game developed for the engine. Much of our experiments to date have focused on a particular game using this engine, Unreal Tournament 2004 (Digital Extremes, 2004).

This paper addresses some of the open problems and questions from our earlier work, and provides architectural refinements and more detailed analysis with a wider variety of data than previously possible. The remainder of this paper is organized as follows: we begin with a discussion of our approach to instrumentation and automation for content analysis. We then describe our implementation and proof of concept work with Epic's Unreal Engine. Next, we discuss our experiences in content analysis experiments on *Unreal Tournament 2004* to date. Finally, we conclude this paper with a summary and a discussion of directions for future work.

Instrumentation for Content Analysis

Software instrumentation is a concept new to video games, but has been used for several years in other types of software to enable the collection of data and the exertion of control over the software. Instrumentation can take many forms; the approach we take is derived from our earlier work in the area of application management (Katchabaw, Howard, Lutfiyya, Marshall, & Bauer, 1999), with refinements and changes to support the specific needs of video game software. The basic premise of our approach is to embed additional code into the execution stream of the game software. By doing this properly, this added code will enable monitoring activities to collect data about the game and the gameplay experiences being generated as the game is played, as well as any control operations that are required in the process.

Instrumentation Architecture

The instrumentation architecture used in our current work is depicted in Figure 1, and discussed in detail in the remainder of this section.

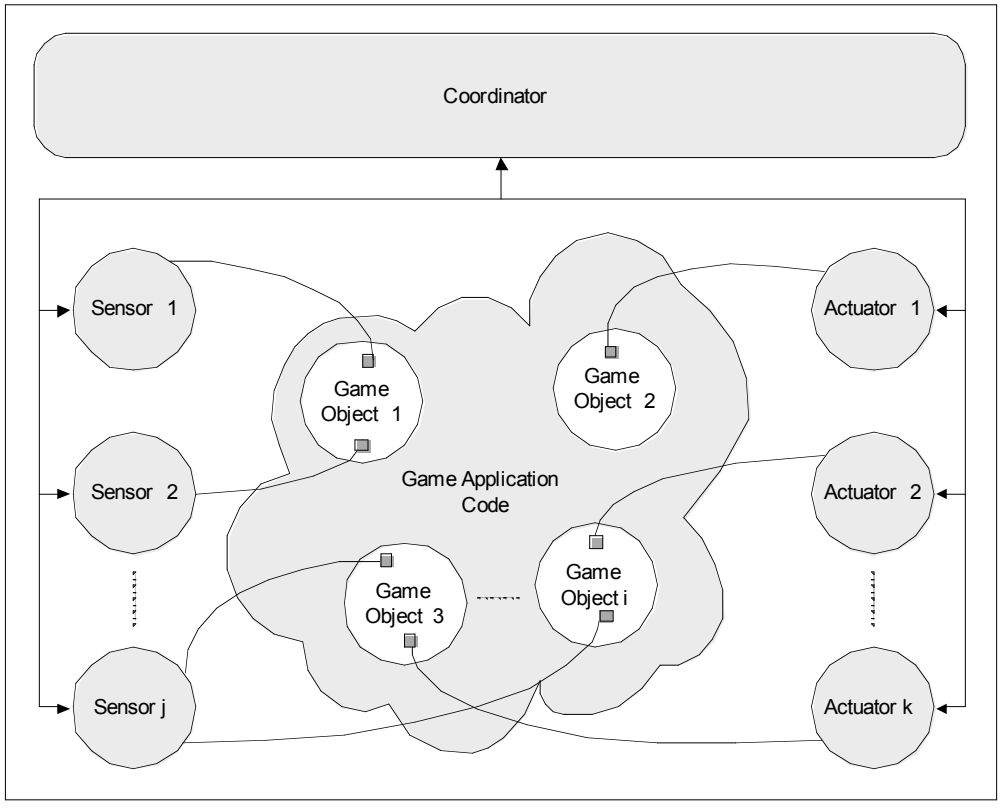


Figure 1: Instrumentation Architecture for Content Analysis

Game Application Code

Game application code refers to the original source code from the game that is being instrumented. It is composed of a collection of objects that work together to deliver the functionality of the game. By gathering data and statistics from the appropriate game objects at the right times, we can conduct an effective quantitative content analysis of the game as it is being played.

Sensors

Sensors are instrumentation components that are used to collect, maintain, and (perhaps) process information to be used in content analysis. Sensors interface with objects in the game application code through probes that are inserted into the game. Such probes are typically macros, function calls, or method invocations that are placed in the execution stream of an object's source code during development, or are event listeners for events emitted by an object as its code executes. Sensors typically reside in the same address space as the game application code, perhaps executing in separate threads. Depending on the game and how it is constructed, however, sensors could theoretically exist in separate processes.

Sensors can be used to collect a wide variety of measurements useful to a content analysis. This includes instances of violence (type of violence, source and target of violence, result of violence), offensive language (what was said, source and target of the language), character demographics (race, age, gender), and so on. Sensors can also collect a variety of game and game world information, including the game being played, the type of game, the level of the game, the time played, and so on.

For flexibility, sensors can also have their behaviours tuned, in some cases at run time. This includes whether they are active or not, what data is being collected, how data is processed, how data is being reported, and so on.

Actuators

Actuators encapsulate functions, which exert control over the instrumented game to change its behaviour in a variety of ways to support the content analysis being conducted. Like sensors, actuators carry out their control operations through interactions with probes at key locations in the game's code. Also, like sensors, actuators are dynamic and can be created or destroyed at run time. Actuators can also be activated and deactivated at run time, like sensors. They perform control operations when requested to do so by the coordinator in place, and may return a result to the coordinator, depending on the operation performed.

Actuators were originally omitted from our earlier work (Bullen, Katchabaw, & Dyer-Witheford, 2006), as it was not believed that such control facilities were necessary for conducting content analysis of video games. This was consistent with traditional content analysis conducted with television or film, where no control or influence over the experience is possible. Such control or influence is, in fact, possible in the realm of video games, since the software elements of video games allow them to be tuned or otherwise manipulated. Consequently, the concept of actuators for content analysis was introduced.

It is expected that actuators can be used in content analysis to assist in the studying of the effects of interactivity and non-linearity on play experiences. Actuators, for example, can force gameplay sequences or outcomes of player actions that might be otherwise difficult to achieve and experience. The use of actuators for such purposes has the potential to enable more thorough analysis than was previously possible. Consequently, development efforts in this area are currently under way.

Coordinator

The coordinator is an instrumentation component that is responsible for directing the content analysis activities occurring within a game. This includes initializing and configuring sensors and actuators, processing reports of collected data and statistics from sensors, issuing commands to actuators, and handling clean-up activities when the game terminates. The coordinator is also the point of contact for tuning behaviour of sensors and actuators, as well as other aspects of content analysis at run-time. Like sensors and actuators, the coordinator also typically resides in the same address space as the game application code, but could be located in a separate process, depending on the game in question.

Instrumentation Operation

When a game instrumented for content analysis is launched, one of its initialization activities before play commences is to create a coordinator to initialize the instrumentation. This, in turn, creates the required sensors and actuators, and configures them to collect data and control the game as required for the content analysis in question.

As the game executes, probes for the sensors will gather the information needed as they are either invoked in the execution stream of the corresponding game objects, or in response to events generated by the game objects, depending on the structuring of the game application code in question. This information is accumulated and processed by the sensors and either reported to the coordinator as it is collected or stored for further processing and reporting in the future. Any such reports received by the coordinator are logged to a file, or otherwise presented or recorded as deemed necessary by whoever is conducting the content analysis.

Furthermore, based on input and feedback from the individuals conducting the content analysis, it might be necessary for the instrumentation coordinator to tune or control the behaviour of the game and the experiences it is generating. In such a case, the coordinator will call upon the appropriate actuators embedded in the game to carry out the necessary actions to adjust the game accordingly. Any results from the actuators will flow back to the coordinator to be reported as necessary.

When the game is completed, or is otherwise terminated, the coordinator flushes out any pending reports and deactivates and destroys all sensors and actuators. At this point the coordinator itself shuts down, and the game is allowed to terminate.

Prototype Implementation

As a proof of concept, we have used our instrumentation framework to instrument Epic's Unreal Engine (Epic Games, 2005) to enable content analysis. We chose to instrument an engine

because engine-level instrumentation enables us to conduct content analysis of all games built on top of that engine without requiring instrumentation on a game-by-game basis. The Unreal Engine is also a popular engine among developers and hobbyists, providing a good collection of games for study in the future. Consequently, the Unreal Engine provides a suitable environment for prototyping and development efforts, allowing us to more easily move to other games and game engines when the time comes.

Instrumenting the Unreal Engine for Content Analysis

Since we were targeting the Unreal Engine in this work, our instrumentation was developed using UnrealScript. UnrealScript has many of the features of a traditional object-oriented language in a programmer-familiar syntax, providing excellent support for extensibility. This allowed us to develop instrumentation for our current needs and gave us flexibility for the future.

While a C or C++ instrumentation library is preferable to provide support across a variety of games and game engines, most game engines used in industry do not provide code-level access to their engines or only do so in a cost-prohibitive fashion, including the Unreal Engine. UnrealScript fortunately provided all the access that was required for our content analysis instrumentation, as we demonstrate below.

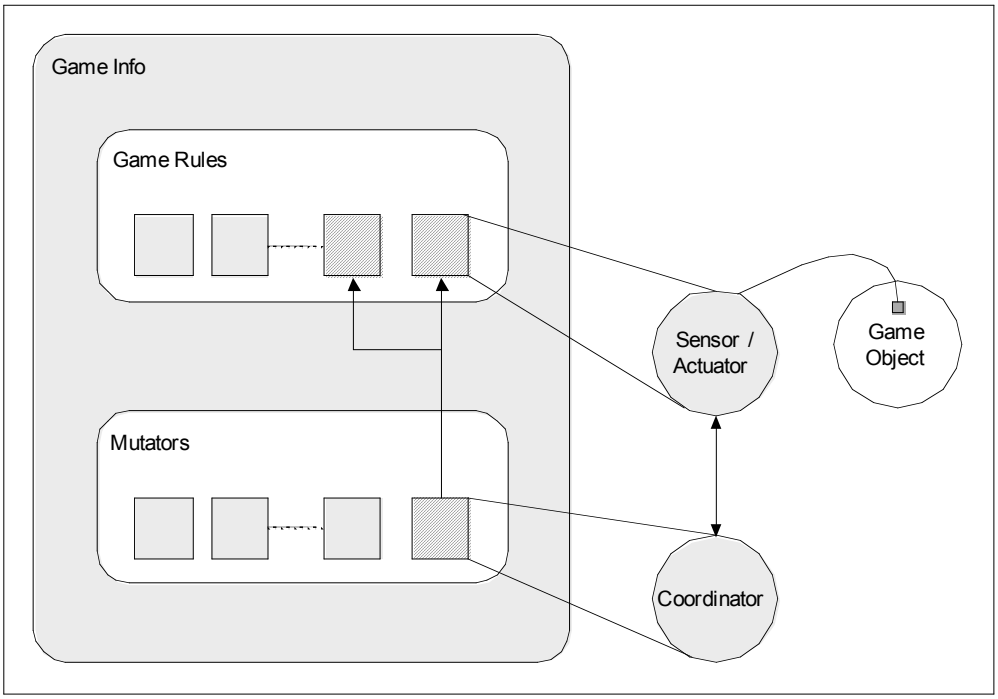


Figure 2: Instrumenting Epic's Unreal Engine for Content Analysis

Adding our instrumentation for content analysis to the Unreal Engine was fairly straightforward, as shown in Figure 2. Each Unreal game type has a Game Info object that defines the game in question. Among other things, this object contains a collection of game rules defining various

aspects of how the game is played, and a collection of mutators. Mutators, in essence, allow modifications to a game and gameplay while keeping the core elements and game rules intact.

Our instrumentation is loaded into a game in the form of a special content analysis mutator. This mutator contains the instrumentation coordinator, as described in the previous section. When loaded, the coordinator in this mutator spawns an appropriate collection of sensors to gather the information required for content analysis and actuators to exert control wherever it is necessary. Each sensor or actuator is contained within a game rule that is appended to the list of game rules contained within the Game Info object by the instrumentation coordinator. In doing so, the sensors are able to access the stream of events generated by the various game objects in the game, and extract the required information to conduct the content analysis. Actuators are also able to access this stream of events, to watch for commands or triggers to call them into action when the individual conducting the content analysis requires an actuator to do something to support the analysis.

For example, suppose we were to conduct a content analysis on a game and were interested in tracking the deaths that occurred within the game. When the content analysis mutator is loaded, the coordinator contained within the mutator creates a new game rule containing a sensor capable of measuring and tracking deaths in the game. This rule is then appended to the list of rules for the game. As the game executes, the sensor in the game rule waits for events indicating that a death has occurred within the game. When a death occurs, the sensor observes the event and updates its internal statistics, perhaps by pulling additional information in from other objects in the process. For example, if we were interested in the cause of death, or the type of death incurred, additional information may be required from the players, weapons, or environment in the game.

Configuring Content Analysis in the Unreal Engine

To support a variety of content analysis, it is important to be able to tailor and configure the content analysis instrumentation in place to tune the data collected by sensors and any commands executed by actuators. Fortunately, the Unreal Engine allows mutators to be selected, configured, and loaded by the user at run-time, which provides a great deal of flexibility for configuring analysis. This allows content analysis to be enabled and disabled dynamically at run-time, and allows the user to tailor and fine tune various elements of the content analysis easily. For example, the user can choose which types of data to collect and not collect, and can tailor various elements of the collection and reporting processes.

For instance, Figure 3 shows the mutator selection screen from *Unreal Tournament 2004* (Digital Extremes, 2004). This game provides an easy-to-use graphical mutator selection mechanism. Figure 4 shows the mutator configuration options available for our content analysis mutator in *Unreal Tournament 2004*. By providing the appropriate UnrealScript, we can easily provide a graphical mechanism to allow users to tailor the content analysis being conducted, as well as provide advanced options for more expert users.

Figure 3: Mutator Selection

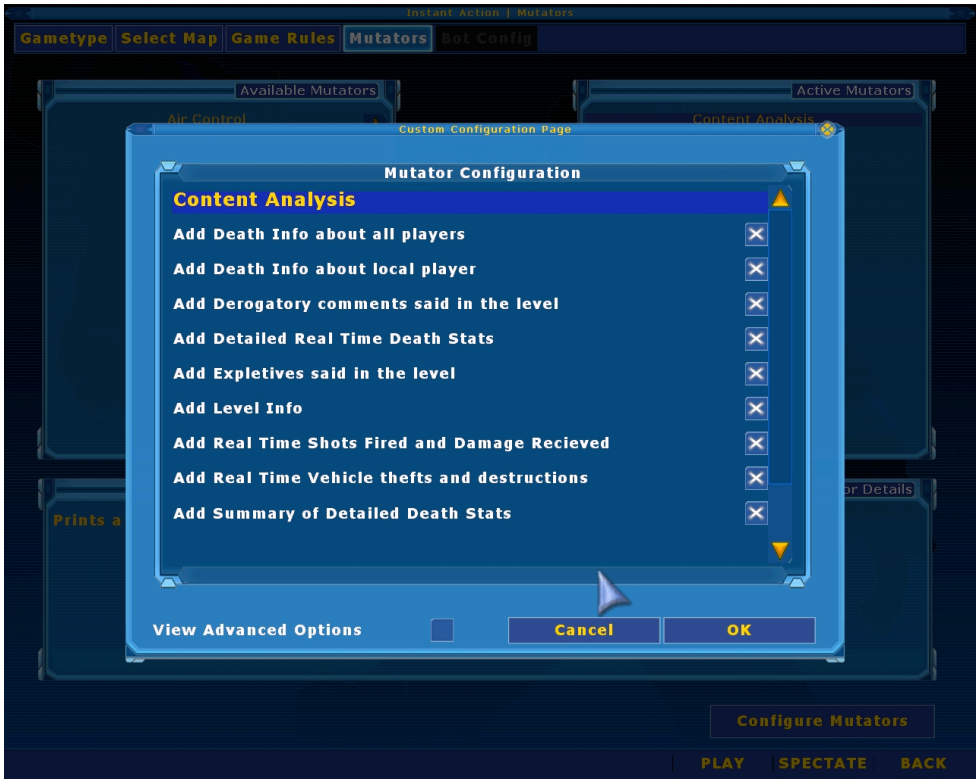


Figure 4: Content Analysis Mutator Configuration

Instrumentation Components Implemented

To date, we have implemented the mutator framework for housing the instrumentation within the Unreal Engine, as well as a coordinator within the mutator for overseeing content analysis instrumentation deployed with the engine. Data collected can either be reported by the coordinate as it is collected and forwarded on by sensors, or in the form of summaries reported when the game is completed or terminated. The method used depends on the needs of the particular content analysis taking place. Unfortunately, the Unreal Engine does not provide a fully functional file access mechanism at the UnrealScript level. However, the Unreal Engine does provide several logging capabilities which are quite sufficient for generating reports of game activities for content analysis. Reports are date and time stamped for easy tracking and organization.

Sensors have been implemented to collect a variety of information required for a thorough content analysis. This includes death of game characters, use of weapons by game characters, use of vehicles in violent or criminal acts, use of offensive language, gender and racial diversity in characters, and a variety of game details such as time played and so on. Depending on configuration, sensors can collect considerable detail on each of the above. For example, consider the tracking of character deaths. Sensors can track not only the incidence of death, but also which character died, what the cause of death was, which other character, vehicle or part of the environment caused the death, the nature of the death (i.e. how violent it was), and whether or not the death was witnessed by a player character.

Actuators are also currently under development for the Unreal Engine for a variety of purposes, mainly to overcome issues of non-linearity and interaction in the game. For example, actuators are under development to artificially force the completion of gameplay elements and objectives to access elements of the game that would have be inaccessible otherwise. This has the potential to enable more thorough content analysis than were feasible before.

Experiences and Discussion

In this section, we describe our initial experiences in using our Unreal-based prototype system for simple content analysis experiments, and discuss observations made in conducting these analyses.

Experiences with Unreal Tournament 2004

To validate our approach to automating content analysis, we needed an Unreal-based game that would use our instrumented Unreal Engine as its foundation. For our purposes, we used *Unreal Tournament 2004* (Digital Extremes, 2004), as it is one of the most popular Unreal-based games, and it was readily available at our disposal. *Unreal Tournament 2004* also ships with development tools for the Unreal Engine, as well as a version of the engine suitable for development purposes, making it an excellent choice for our efforts. *Unreal Tournament 2004* is a first-person shooter game that supports a wide variety of different game types and sets of game rules, individual and team-based games, and single player, multiplayer, and spectator modes of play. In spectator mode, games can be played with no human players, and the game's display is

used to observe the game's progress. Consequently, there are many gameplay options provided within this single game.

The test system used for experimentation was a dual-core 3.0GHz Pentium D system, with 2GB RAM, a 250GB hard drive, and an ATI X1800 graphics accelerator card. The operating system in this case was Microsoft Windows XP SP2. As such, the test system greatly exceeded the recommended system requirements for *Unreal Tournament 2004*. With this experimental environment, we conducted several content analysis experiments using a variety of game configurations. This included the following:

- Standard deathmatch (single player and spectator)
- Team deathmatch (single player and spectator)
- Onslaught (single player and spectator)
- Capture the flag (single player and spectator)

The standard deathmatch game is an individual game, while the other modes were all team-based games, with artificial intelligence-controlled non-player characters filling the rosters of teams. Levels in which the games were played were chosen randomly, and team size and other characteristics as appropriate were set at the levels' default values.

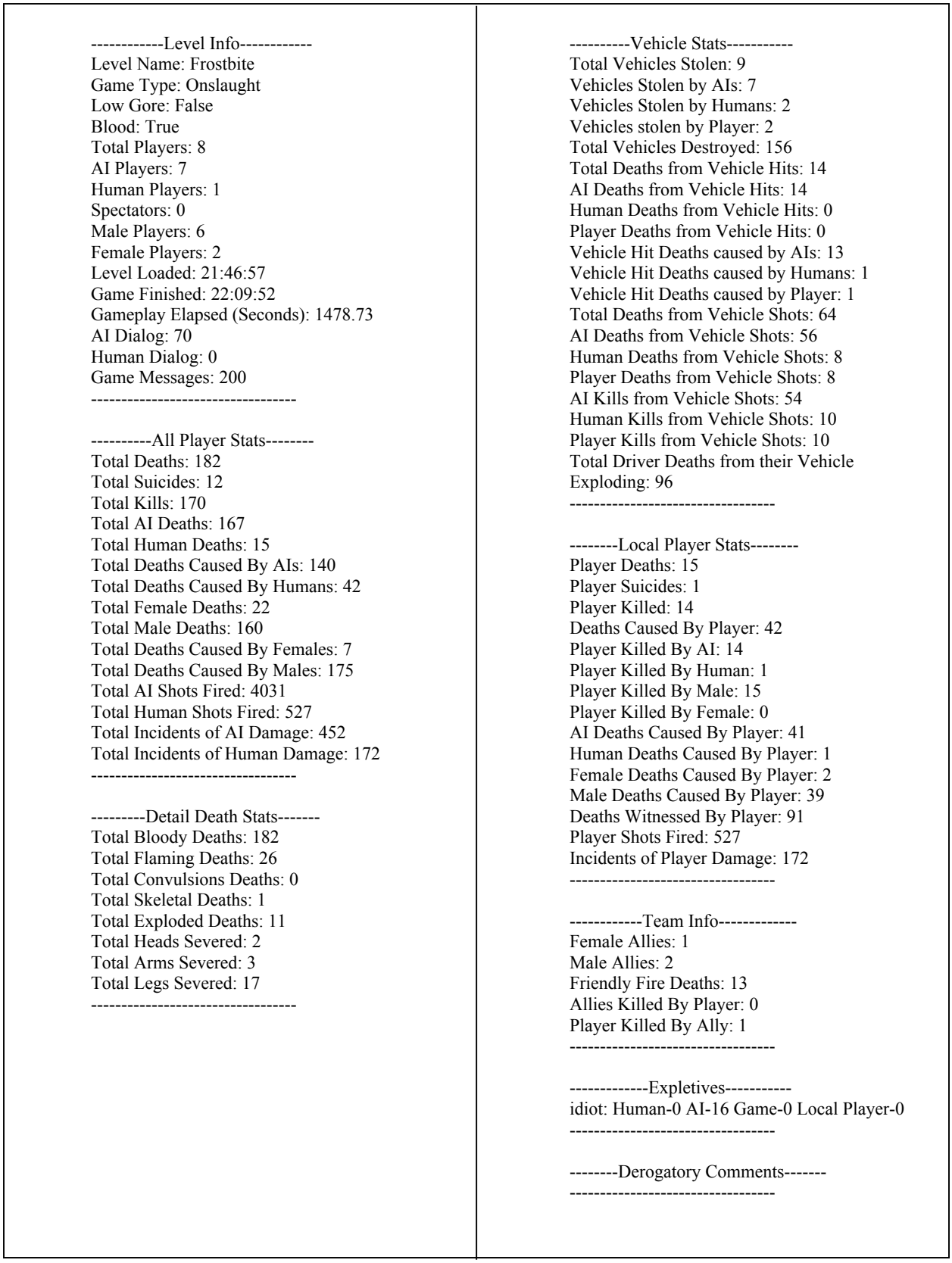
Summary results from one experiment are provided in Figure 5. This summary was taken from an *Unreal Tournament 2004* onslaught game with a single human player and seven artificial intelligence controlled player characters. All content analysis options were enabled, to collect the maximum amount of data possible. Figure 6 shows selected samples of event reports logged during this game. Analysis of the event stream can provide further information about the game in question, and can be matched against video or audio recordings of player experiences to help further analysis.

As demonstrated in Figure 5 and Figure 6, the content analysis instrumentation worked as expected, collecting all of the required data. As a result, the instrumentation appeared to be quite effective in facilitating quantitative content analysis procedures. We are currently investigating experiments to merge these automated quantitative methods with more qualitative methods, to assess the benefits of automation to more comprehensive studies of video games and their content.

Furthermore, this instrumentation was able to provide all required data and statistics with minimal additional work required by the user conducting the content analysis. All that was necessary was to activate the content analysis mutator on its first use, and to collect reports from the generated log file upon completion of the game. After activating the content analysis mutator, it remains active for every game until it is explicitly deactivated.

Further Analysis

Our initial testing and experimentation with our content analysis instrumentation yielded several interesting observations worthy of further discussion and examination.



----------Level Info----------
Level Name: Frostbite
Game Type: Onslaught
Low Gore: False
Blood: True
Total Players: 8
AI Players: 7
Human Players: 1
Spectators: 0
Male Players: 6
Female Players: 2
Level Loaded: 21:46:57
Game Finished: 22:09:52
Gameplay Elapsed (Seconds): 1478.73
AI Dialog: 70
Human Dialog: 0
Game Messages: 200

----------All Player Stats--------
Total Deaths: 182
Total Suicides: 12
Total Kills: 170
Total AI Deaths: 167
Total Human Deaths: 15
Total Deaths Caused By AIs: 140
Total Deaths Caused By Humans: 42
Total Female Deaths: 22
Total Male Deaths: 160
Total Deaths Caused By Females: 7
Total Deaths Caused By Males: 175
Total AI Shots Fired: 4031
Total Human Shots Fired: 527
Total Incidents of AI Damage: 452
Total Incidents of Human Damage: 172

---------Detail Death Stats-------
Total Bloody Deaths: 182
Total Flaming Deaths: 26
Total Convulsions Deaths: 0
Total Skeletal Deaths: 1
Total Exploded Deaths: 11
Total Heads Severed: 2
Total Arms Severed: 3
Total Legs Severed: 17

----------Vehicle Stats----------
Total Vehicles Stolen: 9
Vehicles Stolen by AIs: 7
Vehicles Stolen by Humans: 2
Vehicles stolen by Player: 2
Total Vehicles Destroyed: 156
Total Deaths from Vehicle Hits: 14
AI Deaths from Vehicle Hits: 14
Human Deaths from Vehicle Hits: 0
Player Deaths from Vehicle Hits: 0
Vehicle Hit Deaths caused by AIs: 13
Vehicle Hit Deaths caused by Humans: 1
Vehicle Hit Deaths caused by Player: 1
Total Deaths from Vehicle Shots: 64
AI Deaths from Vehicle Shots: 56
Human Deaths from Vehicle Shots: 8
Player Deaths from Vehicle Shots: 8
AI Kills from Vehicle Shots: 54
Human Kills from Vehicle Shots: 10
Player Kills from Vehicle Shots: 10
Total Driver Deaths from their Vehicle Exploding: 96

--------Local Player Stats--------
Player Deaths: 15
Player Suicides: 1
Player Killed: 14
Deaths Caused By Player: 42
Player Killed By AI: 14
Player Killed By Human: 1
Player Killed By Male: 15
Player Killed By Female: 0
AI Deaths Caused By Player: 41
Human Deaths Caused By Player: 1
Female Deaths Caused By Player: 2
Male Deaths Caused By Player: 39
Deaths Witnessed By Player: 91
Player Shots Fired: 527
Incidents of Player Damage: 172

----------Team Info----------
Female Allies: 1
Male Allies: 2
Friendly Fire Deaths: 13
Allies Killed By Player: 0
Player Killed By Ally: 1

----------Expletives----------
idiot: Human-0 AI-16 Game-0 Local Player-0

--------Derogatory Comments-------

Figure 5: Sample Summary of Content Analysis Data from an *Unreal Tournament 2004* Onslaught Game

----------Player Details----------
Enigma - Female - AI - Robot
Baird - Male - AI - White
Rapier - Male - AI - Robot
Brock - Male - AI - White
Barktooth - Male - AI - White
Lilith - Female - AI - Alien
Skakruk - Male - AI - Alien
Mayhem - Male - Human - White - Local Player

----------Real Time Data----------
23.19: Rapier fired a AVRiL
24.89: Brock fired a Link Gun
25.01: Brock fired a Link Gun
25.10: Brock fired a Link Gun
…
38.65: Unmanned Vehicle Destroyed by Mayhem in a vehicle - Witnessed by Player
39.51: Brock fired a Link Gun
47.50: Baird damaged by Barktooth
48.11: Enigma damaged by Skakruk
48.11: Vehicle Driven by Barktooth Destroyed by Skakruk in a vehicle
48.11: Driver died in Vehicle Explosion Bloody - Witnessed by Player
48.16: Enigma damaged by Skakruk
49.27: Enigma fired a Link Gun
49.44: Enigma fired a Link Gun
49.64: Enigma fired a Link Gun
49.84: Enigma fired a Link Gun
50.04: Enigma fired a Link Gun
50.25: Enigma fired a Link Gun
50.46: Enigma fired a Link Gun
50.56: Enigma fired a Link Gun
50.59: Enigma damaged by Skakruk
50.59: Skakruk in a vehicle Killed Enigma - Kill - Bloody, Flaming *Exploded*
…
105.79: Mayhem damaged by Baird
106.25: Mayhem damaged by Baird
106.25: Baird in a vehicle Killed Mayhem - Kill - Bloody - Witnessed by Player
111.43: Rapier fired a Shock Rifle
112.77: Rapier fired a Shock Rifle
113.46: Rapier fired a Shock Rifle
114.25: Rapier fired a Shock Rifle
114.92: Rapier fired a Shock Rifle
115.20: Enigma damaged by Baird
115.66: Rapier fired a Shock Rifle
116.33: Rapier fired a Shock Rifle
117.07: Rapier fired a Shock Rifle
117.07: Vehicle Driven by Barktooth Destroyed by Barktooth in a vehicle
117.07: Driver died in Vehicle Explosion Bloody
…
255.36: Brock in a vehicle Killed Rapier - Kill - Bloody, Vehicle Hit
260.79: Brock fired a Link Gun
260.96: Brock fired a Link Gun
261.18: Brock fired a Link Gun
261.37: Brock fired a Link Gun
261.58: Brock fired a Link Gun
262.00: Vehicle Driven by Baird Destroyed by Mayhem in a vehicle - Witnessed by Player
262.00: Driver died in Vehicle Explosion Bloody - Witnessed by Player
264.19: Barktooth damaged by Skakruk
264.33: Lilith damaged by Barktooth
264.45: Barktooth damaged by Skakruk
264.57: Lilith damaged by Barktooth
264.89: Lilith damaged by Barktooth
265.12: Lilith damaged by Barktooth
265.27: Barktooth damaged by Lilith
265.27: Vehicle Driven by Barktooth Destroyed by Lilith in a vehicle - Witnessed by Player
265.27: Driver died in Vehicle Explosion Bloody - Witnessed by Player
272.67: Unmanned Vehicle Destroyed by Brock in a vehicle - Witnessed by Player
277.13: Unmanned Vehicle Destroyed by Mayhem in a vehicle - Witnessed by Player
277.13: Unmanned Vehicle Destroyed by Mayhem in a vehicle - Witnessed by Player
277.58: Enigma fired a AVRiL
279.08: Vehicle Driven by Skakruk Destroyed by Enigma
279.27: Barktooth damaged by Lilith
281.06: Enigma fired a Link Gun
281.16: Enigma fired a Link Gun
…
444.59: Vehicle Driven by Skakruk Destroyed by Barktooth in a vehicle - Witnessed by Player
444.59: Driver died in Vehicle Explosion Bloody - Witnessed by Player
444.66: Baird fired a Minigun
444.66: Baird damaged by Mayhem
444.74: Baird fired a Minigun
444.78: Baird fired a Minigun
444.84: Baird fired a Minigun
444.91: Baird damaged by Mayhem
445.26: Vehicle stolen by Baird - Witnessed
447.02: Brock damaged by Barktooth
447.41: Vehicle Driven by Barktooth Destroyed by Baird in a vehicle
447.41: Driver died in Vehicle Explosion Bloody447.41: Unmanned Vehicle Destroyed by Baird in a vehicle
451.26: Vehicle Driven by Baird Destroyed by Mayhem in a vehicle - Witnessed by Player
451.26: Driver died in Vehicle Explosion Bloody - Witnessed by Player
…

Figure 6: Sample Event Stream Log of Content Analysis Data from an *Unreal Tournament 2004* Onslaught Game

Quality of Data

While conducting experimentation with our content analysis instrumentation, we felt it important to verify the accuracy of collected data with more traditional manual procedures using a human observer watching gameplay sessions. In doing so, it was found that the statistics computed by the instrumentation matched those computed using the manual procedures.

Interestingly enough, the statistics computed by the instrumentation appeared to be more complete and more accurate as the pace of the game and positioning of in-game cameras at times made manual procedures error-prone and frustrating, particularly when it came to real-time logging of game events (such as those shown in Figure 6). Instrumentation was also able to capture both on-screen and off-screen activities, and distinguish between the two, which is difficult, if not impossible, to accomplish using manual procedures alone.

Quantity of Data

Another observation deals with the quantity of data collected and how this data is reported. Increasing the amount of data available to a content analysis has the potential to increase its accuracy and the amount of insight that can be obtained from the analysis. Our content analysis instrumentation was found to be able to generate reports with considerable detail, and the elimination of manual collection procedures allows data to be collected from more gameplay sessions than previously possible.

In our earlier work (Bullen, Katchabaw & Dyer-Witheford, 2006), it was noted that increasing the quantity of data handled by instrumentation has the potential to increase processing and storage requirements, as this data must be collected, stored, and reported for use in content analysis. At the time, it was thought that there would be a risk of negative impacts on the performance of the game, particularly when massive quantities of data were not only kept and summarized, but also logged in real-time. Such an occurrence could have negative impacts on the quality of the content analysis, as the analysis itself could be playing a role in the outcome of the games being played.

During our experiments, with all available content analysis metrics being collected and logged in real-time, there was no measurable change in the game's frame rate from when content analysis data was not being collected. Furthermore, the user conducting the content analysis experiments by playing the game noticed no appreciable difference when content analysis data was collected as compared to when it was not.

Consequently, while the quantity and frequency of data collection has the potential to cause performance issues with the game being analyzed in theory, our experiments found no such issues in practice, even with a very large amount of data being collected in real-time as the game was played.

Partial versus Fully Automated Content Analysis

Another interesting observation came when comparing partially automated content analysis to fully automated analysis. A partially automated analysis requires a human player to drive the game while the embedded instrumentation handles the data collection and reporting

activities, whereas a fully automated analysis requires no human player, with the game essentially driving itself using artificial intelligence-controlled non-player characters.

Since *Unreal Tournament 2004* supports a spectator mode in its game sessions, it is possible to conduct a fully automated content analysis on the game, simply by having artificial intelligence-controlled non-player characters play the game by themselves. Unfortunately, these games can take significantly longer than games involving human players, as the non-player characters tend to be less effective at achieving victory than human players. Also, since the skill level of non-player characters is more balanced, the kills in a game can be more evenly distributed in the absence of a dominant human player, requiring more kills in total to end a game.

For example, consider the onslaught game whose summary and event log are shown in Figures 5 and 6 respectively. The human player clearly dominated this game, scoring more than a quarter of the total kills in the entire game, and quickly bringing the game to an end in reaching the victory condition of the game. When played in spectator mode with the same number of non-player characters in the same level, the game took nearly three times as long to complete on average, with a much more balanced kill distribution between the teams in the game. With the human player no longer dominating in the game, the game results were substantially different. Comparisons between other types of games showed similar results.

This indicates that the nature of data collected during a partially automated content analysis might differ significantly from a fully automated analysis. Since a partially automated analysis involving a human player is likely a more accurate reflection of an actual gameplay experience than a fully automated analysis, this raises questions about the suitability and validity of fully automated analysis. However, since a fully automated analysis removes the need for human interaction with the game, this kind of analysis is still attractive as it is less resource intensive, allows data to be collected from more game sessions, and removes bias and unwanted effects introduced by the human players of the game.

In the end, the approach taken likely depends on the questions being studied as part of the analysis being conducted. If the goal of the study is to examine the player's experience, this can likely be best accomplished through a partially automated content analysis, as a human player is still involved in the process. On the other hand, if the goal of the study is to examine the game itself and the expression and messages made by the game, this might be best accomplished through a fully automated analysis, as any player bias is removed and it might be possible to obtain a clearer view of the game as a result. Clearly, this issue requires further study.

Conclusions and Future Work

To provide better supports for content analysis of video games, this paper discusses an approach to partially or even fully automating analysis procedures using a collection of instrumentation components embedded within the game software itself. By automating content analysis, we can increase the accuracy and scope of studies of games, while at the same time reducing the time and costs of such studies.

Based on our approach, a prototype implementation was developed to enable automated content analysis of games based on Epic's Unreal Engine. Using this prototype, several experiments have been conducted with *Unreal Tournament 2004*, demonstrating the effectiveness of our approach, and showing great promise for future efforts in this area. As such, there are many possible directions for future work in this area, including the following:

- Based on the success of initial content analysis experimentation, as discussed in the previous section, more thorough and detailed content analysis should now be conducted on *Unreal Tournament 2004*, combining quantitative data collected through our instrumented engine with more qualitative observations.

- Content analysis experiments should be expanded to include more Unreal-based games as well, to provide further validation of our instrumentation.

- With success in instrumenting Epic's Unreal Engine, we should investigate the instrumentation of other game engines to enable automated content analysis in a wider collection of games.

- As mentioned in the previous section, additional testing and experimentation are required to study the advantages and disadvantages of partially automated content analysis compared to fully automated analysis.

References

Bogost, I. (2006). *Unit operations: An approach to videogame criticism*. Cambridge, MA: MIT Press.

Brand, J. and Knight, K. (2003, November). *The diverse worlds of computer games: A content analysis of spaces, populations, styles and narratives*. Paper presented at the DiGRA 2003: Level Up, Utrecht, The Netherlands.

Bullen, T., Katchabaw, M., & Dyer-Witheford, N. (2006, September). *Instrumentation of video game software to support automated content analysis*. Paper presented at GameOn North America 2006, Monterey, California.

Digital Extremes. (2004). *Unreal Tournament 2004* – Editor's Choice [Computer Software]. London, ON: Epic Games.

Epic Games. (2005). *Unreal Engine 2, Patch-level 3369*. Rockville, MD: Epic Games.

Grimes, S. (2003, November). *You shoot like a girl: The female protagonist in action-adventure games*. Paper presented at DiGRA 2003: Level Up Utrecht, The Netherlands.

Heintz-Knowles, K., Henderson, J., Glaubke, C., Miller, P., Parker, M. and Espejo, E. (2001, December). *Fair play: Violence, gender and race in video games*. Oakland, CA: Children NOW.

Holsti, O. (1969). *Content analysis for the social sciences and humanities*. Reading, MA: Addison-Wesley.

Katchabaw, M., Howard, S., Lutfiyya, H., Marshall, A., and Bauer, M. (1999). Making distributed applications manageable through instrumentation. *Journal of Systems and Software*, 45(2), 81-97.

Newman, J. (2004). *Videogames*. New York: Routledge.