ABSTRACT

DOMINGUEZ, MICHAEL A. Automatic Identification and Generation of Highlight Cinematics for 3D Games. (Under the direction of Dr. R. Michael Young).

Online multiplayer gaming has emerged as a popular form of entertainment. During these games, the players' main focus is usually placed on achieving the objectives that must be completed to win the game. While these tasks may be of the primary interest to the players, over the course of the game their interactions may result in interesting narratives that go unnoticed. This may be due to the imperfect information that a player has access to or as a result of their attention being directed towards accomplishing the goals of the game. This thesis presents Afterthought, a system that will allow players to view these emergent narratives after completing their gameplay session. The tool accomplishes this through logging the actions that occur during the play of the game, analyzing the log and retrieving interesting narratives, generating the cinematic discourse for visualization, rendering the videos of the narratives, and finally uploading the videos to a video sharing site so that they are easily viewable by all participants. This thesis concludes with preliminary human subjects evaluation of the system's effectiveness.

Automatic Identification and Generation of Highlight
Cinematics for 3D Games

by
Michael A. Dominguez

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fullfillment of the
requirements for the Degree of
Master of Science

Computer Science

Raleigh, North Carolina

2010

APPROVED BY:

Dr. Benjamin Watson

Dr. Robert St. Amant

Dr. R. Michael Young
Chair of Advisory Committee

DEDICATION

To my parents.

BIOGRAPHY

Michael Dominguez was born in Passaic, NJ and grew up in Tannersville, PA. After graduating from Lafayette College in 2007 with a Bachelor of Science in Computer Science, he joined the Liquid Narrative group at North Carolina State University to pursue research into automated camera control and machinima generation.

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. R. Michael Young, for his support and guidance. I would also like to thank my committee members, Dr. Robert St. Amant and Dr. Benjamin Watson.

Thanks to all of the Liquid Narrative group members, and especially to Stephen Roller for his contributions to Afterthought.

Finally, I would like to thank my parents, Dennis and Nancy, and my brothers, David and Philip.

TABLE OF CONTENTS

LIST OF TABLES . . . . . **vii**

LIST OF FIGURES . . . . . **viii**

1 Introduction **1**
1.1 Thesis Organization 4

2 Related Work **5**
2.1 Commercial Games 5
2.2 Research Work 7
2.2.1 Game Log Analysis 7
2.2.2 Machinima Generation 9
2.2.3 Automated Camera Control 11

3 Description of Afterthought **15**
3.1 Overview 15
3.2 Game Requirements 17
3.3 Game Server and Log Dispatcher 17
3.3.1 Unreal Tournament 3 Mod 18
3.4 Narrative Pattern Matcher 19
3.5 Cinematic Discourse Generator 26
3.5.1 Resolving Timing Conflicts Between Camera Actions 27
3.5.2 Adding Slow Motion Effects 32
3.5.3 Adding Recording Actions 32
3.5.4 Loading Camera Modifiers 33
3.6 The Renderer 33
3.6.1 Unreal Tournament 3 Mod 38

4 Preliminary Evaluation **40**
4.1 Participants, Materials, and Apparatus 40
4.2 Procedure 41
4.2.1 Deciding What Narratives To Film 41
4.3 Results 42
4.4 Discussion 45

5 Conclusions and Future Work **47**
5.1 Future Work 47

Bibliography **50**

Appendices **54**
Appendix A 55
Appendix B 77

LIST OF TABLES

Table 3.1 All action types logged in the modification to *UT3*'s Capture the Flag game type. 18

Table 4.1 The possible viewing orders assigned to the subjects. Narrative 1 and Narrative 2 were filmed both with our custom camerawork (A) and without (B). 42

Table 4.2 The mean scores given to the specified qualities in each group. 43

Table 4.3 The mean of the difference between the scores given by each participant for the video of Type A and the video of Type B of the same narrative. 43

Table 4.4 The standard deviation of the difference between the scores given by each participant for the video of Type A and the video of Type B of the same narrative. 44

Table 4.5 T-statistic for the difference between the scores given by each participant for the video of Type A and the video of Type B of the same narrative. 44

Table 4.6 Responses to the question, "When you were playing the game, did you witness or participate in any of the depicted events in the video?" 44

Table 4.7 Responses to the question, "Did you describe part or all of the events in this video in your gameplay survey?" 45

LIST OF FIGURES

Figure 1.1 Both frames show the same character in the same environment, however by altering the camera a filmmaker can change the viewer's perception of the character and the scene. 2

Figure 2.1 The default third person viewpoint for replays in *Unreal Tournament 3*. The viewer is able to rotate the camera around the targeted player using the mouse. . . 6

Figure 3.1 Afterthought's architecture. 16

Figure 3.2 An example pattern for use with the Capture the Flag game type in *UT3* that describes a player's unsuccessful attempt at capturing the other team's flag. In this pattern, a player picks up the opponent's flag, leaves the area near the opponent's base, and enters the area near their own team's base. Then, an opponent enters the area near the base, kills the player, and the flag is returned. 20

Figure 3.3 An example pattern for use with the Capture the Flag game type in *UT3* that describes a player picking up his opponent's flag, getting zero or more kills, and then scoring. 21

Figure 3.4 An example pattern for use with the Capture the Flag game type in *UT3*. Player A goes to get flag, but it's not there. Player B returns it, then Player A picks it up and scores. 22

Figure 3.5 A illustration of the internal representation of the narrative pattern specified in Figure 3.4. At each transition, the `withintime` check is performed to see if the specified time has elapsed - if it has, the pattern fails. `dontmatch` ensures that patterns we do not want to match (not shown in illustration) have not occurred between the `dontmatch`'s start and check. 23

Figure 3.6 An example set of narratives sent from the narrative pattern matcher to the cinematic discourse generator. These are the narratives that would be generated if narrative pattern matcher tried to match the pattern in Figure 3.3 with the actions specified in the log messages from Figure A.2. 25

Figure 3.7 The steps taken to generate a camera event. 26

Figure 3.8 The example camera event after loading the camera action series for each action in its associated narrative. Each box represents a single camera action and is

labeled with text that indicates its status as a primary or secondary camera action, its associated type of action from the narrative, and its starting and ending time. The smaller boxes below each camera action represent array indices. Time proceeds from left to right, although timing conflicts exist in this example, as shown in Figure 3.9. 28

Figure 3.9 Illustrating how the camera actions shown in Figure 3.8 overlap with each other. The overlap indicates that there are timing conflicts that need to be resolved (boxes not to scale of timeline). 28

Figure 3.10 The algorithm used to resolve timing conflicts between primary camera actions. 28

Figure 3.11 Screenshots illustrating a transition. The first screenshot is a close-up, while the final is a long shot behind the actor. The frames in between show the transition taking place from one to the other. 30

Figure 3.12 The algorithm used to resolve timing conflicts regarding the spawn times of actors for secondary camera actions. 30

Figure 3.13 The algorithm used to resolve timing conflicts regarding secondary camera actions. N = number of camera actions currently in the camera event. 31

Figure 3.14 An example camera action for a long shot following behind the player. The camera modifiers used will locate the camera in back of the target at a long distance, and will keep its rotation facing the same direction as the target. In addition, it will use a normal field of view. 34

Figure 3.15 CamMod-LocationRelative-BackLong: A camera modifier that sets the location of the camera based on a vector offset relative to a specified target actor. In this case, the camera will be placed directly behind the target actor at distance of 100 units. The start time, end time, and target will be specified when the camera modifier is instantiated. The direction angles are specified in degrees, and the distance is specified units particular to the Unreal Engine. 35

Figure 3.16 CamMod-RotationRelative-Forward: A camera modifier that sets the rotation of the camera to be equal to that of its target actor. The start time, end time, and target actor will be specified when the camera modifier is instantiated. The direction angles are specified in degrees, and the distance is specified in units particular to the Unreal Engine. 35

Figure 3.17 CamMod-FOV-Normal: A camera modifier that sets the field of view to be its normal value. The start time and end time will be specified when the camera modifier is instantiated. The FOV angle is specified in degrees. 36

Figure 3.18 Screenshots from the video generated using the camera event from Figure A.6 with the game session described in the log messages in Figure A.2. 37

Figure 3.19 An example AviSynth script generated by the renderer to concatenate the videos comprising a narrative. 38

Figure 5.1 A screenshot from the TV series *24* illustrating the use of multiple viewpoints, which allows the audience to witness simultaneously occurring actions that take place in different locations. 48

Figure A.2 An example set of log messages sent from a Capture the Flag match in *Unreal Tournament 3*. The timestamps refer to the number of seconds that have passed since the start of the match. Some messages from this game session were removed for conciseness. Continued on next page. 56

Figure A.2 (Continued) An example set of log messages sent from a Capture the Flag match in *Unreal Tournament 3*. Continued on next page. 57

Figure A.2 (Continued) An example set of log messages sent from a Capture the Flag match in *Unreal Tournament 3*. 58

Figure A.3 The camera action series used for the flag pickup action with the player as the focus. Continued on next page. 59

Figure A.3 (Continued) The camera action series used for the flag pickup action with the player as the focus. Continued on next page. 60

Figure A.3 (Continued) The camera action series used for the flag pickup action with the player as the focus. 61

Figure A.4 The camera action series used for the kill action with the killer as the focus. Continued on next page. 62

Figure A.4 (Continued) The camera action series used for the kill action with the killer as the focus. Continued on next page. 63

Figure A.4 (Continued) The camera action series used for the kill action with the killer as the focus. 64

Figure A.5 The camera action series used for the flag score action with the player as the focus. Continued on next page. 65

Figure A.5 (Continued) The camera action series used for the flag score action with the player as the focus. Continued on next page. 66

Figure A.5 (Continued) The camera action series used for the flag score action with the player as the focus. 67

Figure A.6 The camera event generated by using the camera action series show in Figure A.3, A.4, and A.5 with the narrative with ID = 1 in Figure 3.6 and the log messages from A.2. Continued on next page. 68

Figure A.6 (Continued) Example camera event. Continued on next page. 69

Figure A.6 (Continued) Example camera event. Continued on next page. 70

Figure A.6 (Continued) Example camera event. Continued on next page. 71

Figure A.6 (Continued) Example camera event. Continued on next page. 72

Figure A.6 (Continued) Example camera event. Continued on next page. 73

Figure A.6 (Continued) Example camera event. Continued on next page. 74

Figure A.6 (Continued) Example camera event. Continued on next page. 75

Figure A.6 (Continued) Example camera event. 76

Figure B.1 The first page of the reference sheet given to the participants in the preliminary evaluation, describing the game's rules and controls. 78

Figure B.2 The second page of the reference sheet given to the participants in the preliminary evaluation, describing the game's heads up display. 79

Figure B.3 The survey completed by each participant in the preliminary evaluation as they watched the replay of the match they just played. 80

Figure B.4 The survey completed by each participant in the preliminary evaluation for each video generated from their gameplay session. 81

Chapter 1

Introduction

Video and computer games have emerged as a popular form of entertainment. Many games feature an online multiplayer component in which players can interact with each other. Over the course of an online multiplayer gameplay session, interesting narratives can emerge through the interactions between these players. In many cases, these may go unnoticed by the participants, possibly due to their primary focus of completing the game's objectives or their lack of complete knowledge of the dynamic state of the virtual world. In order to experience these narratives, players need a method of reviewing past actions that took place during gameplay.

Many commercial games provide methods to analyze past moments of gameplay. A typical player-controlled replay system gives users free control of the viewpoint, putting the burden on the player to find the most interesting moments and to position the camera such that they can adequately see the action taking place. Some replay systems give users the ability to pause, rewind, and fast forward through the action [1]. Others include more advanced features such as clipping a replay to a certain segment and the integration of a mechanism to share clips with other players [2]. Some games can automatically generate a video showcasing a highlight, however they may be limited to featuring only the most obviously important moments, such as capturing a flag or scoring a goal [3].

While these existing systems provide many useful features, they lack the ability to both intelligently recognize complex interactions between players during gameplay and subsequently effectively create a video illustrating them. For example, perhaps the following scenario occurs in a first person shooter. Player A enters a courtyard and is killed by Player

B, a sniper in a bell tower. Player A respawns and enters the courtyard again, but is killed by a grenade thrown by his teammate, Player C. Player A respawns and enters the courtyard once more, and is killed by Player D's shotgun. Players B, C, and D may not be aware of Player A's repeated misfortune and could find this series of events humorous. An effective way of presenting these events to the players would be to create a video that shows shot sequences of Player B killing Player A, followed by Player C killing player A, followed by Player D killing Player A.

When a viewer experiences a visual narrative, the way that the actions of the story are presented to them can dramatically impact their perception and understanding of events. Filmmakers have taken advantage of this since the advent of the medium in order to communicate details about the story to their audience. A number of works have been written describing the typical methods used for effective filmmaking (e.g. [4, 5]).

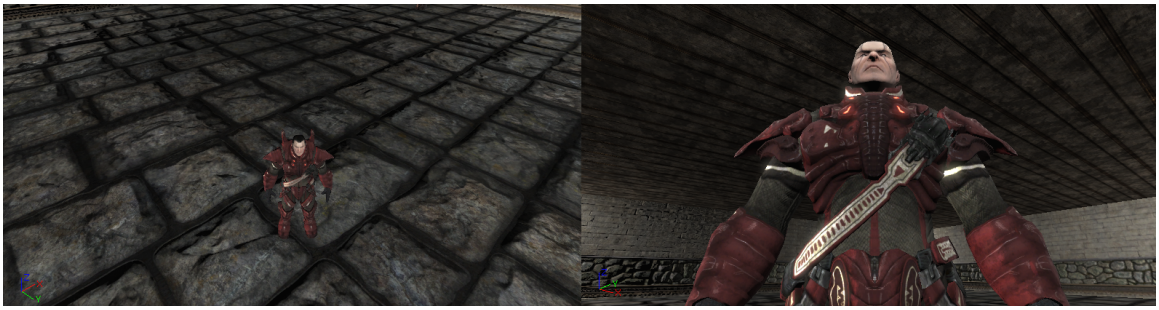


Figure 1.1: Both frames show the same character in the same environment, however by altering the camera a filmmaker can change the viewer's perception of the character and the scene.

See Figure 1.1 for an example of how changing a viewer's perspective of a scene can alter their perception of a character. In both of the images, the man and the environment are exactly the same; however the perspective of the viewer changes dramatically. In the first frame, the man is shown at a distance, and he is viewed from a high vantage point. This leads the audience to have little emotional attachment to the character, and they perceive him as weak due to his relatively small size in the frame. In the second frame, the same character is in the same position, but the location of the camera has changed. In this instance, the camera has moved closer and is angled up towards the man. Now his figure appears imposing and menacing. Techniques such as this are used to affect an audience's

emotions and their understanding of the events taking place. By allowing for more expressive cinematography to be used during replays of past gameplay, the events that take place can be more effectively communicated to the audience. When films are created within a real-time virtual environment, as opposed to live-action or using specialized animation software, they are called *machinima* [6].

This thesis presents Afterthought, a system designed to create videos that depict emergent narratives that occur during gameplay. The problems addressed in the creation of this system include:

- Logging all of the significant actions that take place in a gameplay session.
- Recognizing the narratives that occur in game logs.
- Determining the most effective way of filming these narratives.
- Rendering the video file.

To achieve these goals, the components of Afterthought include:

- The modified game to be used with the system.
- The log dispatcher.
- The narrative pattern matcher.
- The cinematic discourse generator.
- The video renderer.

As a multiplayer game takes place, actions performed by the players will trigger log messages that are sent from the modified game to the log dispatcher. The log dispatcher then forwards the messages to the narrative pattern matcher and to the cinematic discourse generator. As it receives these log messages, the narrative pattern matcher identifies every sequence of gameplay actions that match the narratives it has been instructed to search for. At the conclusion of the gameplay session, the narrative pattern matcher sends these narratives to the cinematic discourse generator. The cinematic discourse generator then determines which narratives to film and how they should be shot and edited. These instructions for filming are then sent to the video renderer, which records the videos and

uploads them to a video sharing web site. This system has been fully implemented for the Capture the Flag game type in the first-person shooter *Unreal Tournament 3* [7].

1.1 Thesis Organization

Chapter 2 describes related research work in regards to game log analysis, the creation of machinima, and automated camera control. In addition, it includes an overview of the different capabilities of replay systems in commercial games. Next, Chapter 3 offers a description of Afterthought's architecture. Chapter 4 presents a preliminary evaluation of the system, and Chapter 5 offers conclusions and describes future work that can build off the work introduced in this thesis.

Chapter 2

Related Work

This thesis builds upon a variety of previous work. Section 2.1 discusses the replay capabilities found in commercial games. Section 2.2 reviews previous research work into game log analysis, machinima generation, and automated camera control.

2.1 Commercial Games

In regards to tracking the interactions between characters, some games perform analysis to find exceptional occurrences. For example, in *Call of Duty: Modern Warfare 2* [8], players receive medals and accolades in response to the actions they perform in the game. A player may receive a medal for killing an enemy who just killed a teammate, or get an accolade for killing the entire enemy team without dying. The game does not attempt to generate any videos highlighting these events.

Many games already include features that allow players to review their past gameplay sessions. However, these systems do not reason about the filming of complex interactions between players.

Some games, such as *Unreal Tournament 3*, allow players to save a "demo file" that contains the data regarding all of the actions taken by players during a gameplay session. The players' actions are pantomimed using the data to provide an accurate replay of what took place. Typically when watching a replay, users can either focus their viewpoint on a single player, as shown in Figure 2.1, or enable the "free camera" mode in which they have manual control and can float freely around the game world. This allows them unrestricted

access to view what had occurred during the game. However, to find interesting gameplay moments they must peruse the entire replay of the match, and even then they may choose viewpoints that do not make the narratives apparent. Some games, such as *Halo 3* [2], feature an integrated system for sharing these replays with other players.

Figure 2.1: The default third person viewpoint for replays in *Unreal Tournament 3*. The viewer is able to rotate the camera around the targeted player using the mouse.

Other games allow players to view instant replays of recent gameplay. These may be shown automatically by the game after it recognizes an important occurrence, or a player may choose to pause their gameplay in order to review some event. Such functionality is present in nearly every major sports game. For example, in the soccer game *Mario Strikers Charged* [3], scoring a goal causes the game to show an instant replay of the event. This concept of instant replays has crossed over into other genres as well, such as the "KillCam" in the *Call of Duty* [9] series. After a player is killed, they are shown an instant replay from the perspective of the player who killed them.

The work presented in this thesis aims to improve upon the capabilities already present in commercial games.

2.2 Research Work

Related work includes research into game log analysis, machinima generation, and automated camera control.

2.2.1 Game Log Analysis

A number of systems have been developed to analyze the logs of gameplay for both virtual and live-action games.

Cheong et. al. developed a system called ViGLS (Visualization of Game Log Summaries) that aimed to automatically create video summaries of past gameplay [10]. One disadvantage of this system is its method of replaying the actions for recording. They make use of a plan structure to reenact events that occurred during gameplay. While this ensures that the significant actions for this narrative will be shown in the replay, the visual representation may not be identical to what actually occurred in game, especially if narratively insignificant actions occurred in close proximity. In contrast, the Afterthought system makes use of the demo recording capabilities of the Unreal Engine to ensure an exact visual representation. In addition, ViGLS films an action by starting its corresponding camera shot at the beginning of the execution of the action, and remains using this camera shot until the next action begins executing. This method is ineffective when significant actions take place in rapid succession (fast camera cuts may disorient the viewer) or occur very far apart (resulting in a long stretch of possibly uninteresting video). Afterthought takes measures to address these timing issues.

Friedman et. al. also worked on creating video summarizations based on logs of activity in virtual environments [11]. The actions in a log are compared against predefined interesting actions, characters, and series of actions. In the case of these predefined series of actions, called routines, the system does not look for an exact match; rather, it looks for certain deviations from a routine and deems these as interesting events. For the movie construction, they address the issue of disruptive camera cuts, which can occur between shots that happen at disparate times but are shown consecutively and feature the same characters and location. To solve this problem, actors that partake in two consecutive tagged actions are afforded an “ease-in phase” into an action in order to remove any potential distractions. They also discuss, but do not implement, the use of dissolves, which are utilized

by Afterthought as transitions between shots in which time has elapsed between the actions they feature.

Halper and Masuch created a system that can extract and evaluate action scenes in computer games [12]. They introduced an evaluation function used to determine the interest level at a particular time during gameplay. The interest level depends on a number of in-game variables, including changes in a player's health, amount of ammunition, rate of fire, etc. Using this interest function, they divide the game session's timeline into distinct scenes of action. Their goal was to utilize these techniques for a spectator mode, in which outside observers view the gameplay live, as well as for gameplay summarization. For the live coverage problem, they discuss a number of methods used to most effectively place a spectator's viewpoint at each moment in time. As to gameplay summarization, they proposed using a modified version of the CAMPLAN system by Halper et. al [13] in order to determine the most effective camera placement for taking a representative screenshots of a scene. The screenshots from the most important scenes could then be presented as a summary of the events.

Chan et. al. describe a mixed-initiative system used to generate comic panels summarizing gameplay [14]. A log is recorded of significant user actions, and screenshots are taken when these events occur as well as at specified regular intervals. These screenshots are presented to the user in the form of comic panels with embellishments such as speech bubbles and onomatopoeic sounds. Users can then customize these generated panels to their liking. A number of limitations exist with this approach. First, the pictorial representations used in the comic are limited to the viewpoint used by the player at the time an action takes place. Narratively significant actions may have taken place away from the player's point of view and would never be present in the final comic strip. In addition, the player's point of view may not be the most effective viewpoint for depicting the actions that they do witness. In contrast, the videos that Afterthought creates do not depend on the point of view of any player during the game and is able to capture actions taking place anywhere in the virtual environment. An earlier comic generation system created by Shamir et. al. [15] used the concept of idioms, which in this case is defined as a common way to depict each type of action in comic form, to determine how to present interactions between players. This issue is addressed in a similar manner by Afterthought.

A number of works have examined the process of automatically generating replays

for activities in the real world, particularly for sporting events. Wang et al. developed a system for generating highlights of a soccer match to be inserted during the broadcast of the game [16]. They perform audio and video analysis on a camera feed to detect when interesting moments occur, then insert replays of these moments over uneventful portions of the game. Xu and Chua describe a system that uses external knowledge, such as game logs available on the Internet, to supplement data they gather from a video source [17].

Drawbacks are evident when comparing replay generation for real-world events with those made for virtual worlds. Typically, in the real world, the data regarding what is taking place must be extracted from a video source or parsed from a manually created game log. In contrast, a virtual world can be modified such that it is instructed to log actions immediately as they are taking place. This removes any reliance on human input or on an algorithm to record what actions have occurred. A second drawback is the limitations on how the action can be presented after detecting the significant moments. In the real world, a limited number of camera viewpoints are available to reconstruct an event, whereas in a virtual world whatever viewpoints will be most effective can be used.

The usefulness of logging game events extends beyond creating summarizations or highlights. Nacke et. al. describe creating modifications to *Half-Life 2* [18] using the Source SDK in order to easily correlate game events with pyschophysiological data taken from instruments monitoring the player [19]. When a player performs an in-game action, such as firing a weapon or entering a specified area, a log message is simultaneously written to a file and sent through a parallel port to pyschophysiological recording hardware. By using an automated logging system, the burden of manually scoring events from an experiment is thus lifted from the researchers.

In addition, there has been a number of instances where machine learning has been applied to game logs in order to develop intelligent agents [20, 21].

2.2.2 Machinima Generation

Machinima movies can feature events that occur during the normal play of a game, or they can be specially crafted by filmmakers. In regards to the former, many players choose to share machinima of their gameplay to showcase their ability to quickly complete an in-game task or to show their mastery of a skill. For the latter, filmmakers manually choreograph the actors and camera movements in order to create a movie. A

virtual camera can jump to a new location instantly, allowing for the real-time cutting of a film if the filmmaker desires. This is somewhat akin to a sporting event that is aired live, however in the virtual world no restrictions are imposed as to where it is possible to place a camera. Similarly, in a virtual world a camera has a freer range of movement, allowing it to easily achieve complex shots that would be extremely costly in the real world. The remainder of this section details selected research work related to the creation of machinima.

SpaceCam [22] is a cinematography and editing system that allows filmmakers to create movies of *Unreal Tournament 2004* [23] gameplay. Similar to the system described in this thesis, it utilizes a modification to the game in order to obtain a written log and makes use of a demo file recording for later playback. During gameplay a stream of player positioning data is sent to a simultaneously executing Java application. After the game has completed, the Java application renders a visualization of the path that the players have traveled through the environment. Cameras can then be placed and manipulated using a tablet interface. This project differs from the work presented here in that it aims to have a user, rather than the system, decide the important moments that need to be filmed and also to choose the correct camera placements.

Elson and Riedl put forth a system, Cambot [24], to automate the cinematography in machinima productions. The system, designed to select appropriate camera angles for a given script, has its process modeled after real-world film production. The authors note several elements typical to movie scripts - the specification of where a scene takes place, the blocking (positioning) of the actors in the scene, and any restrictions of the camera's viewpoint. Even with these instructions, a director still has a great deal of freedom as to how he wishes to meet the constraints of a film. Cambot is able to explore these options and determine the optimal way to present the scene.

Cambot takes as input a set, which is an environment annotated with occlusion-free stages, and a symbolically encoded script that may include constraints as to where and how a scene should be filmed. These constraints are matched against hand-authored facets, which include stages, blockings (placements of characters on a stage), and camera shots. To create the scene, Cambot superimposes stages, blockings, and shots together. There is a many-to-many mapping, but not total compatibility among them. Cambot's search algorithm forms a reel for each scene and then instructs the visualization engine to act out and record the film.

By exerting strict control over the stage and the blocking of the characters, the system avoids having to concern itself with complex occlusion detection and other difficulties. The following section deals with automated camera control in which the environment is not as structured.

2.2.3 Automated Camera Control

In an unpredictable environment where multiple actions can occur simultaneously, a constraint-based approach can be an effective method of filming the events.

Developed by Drucker and Zeltzer, CamDroid [25] utilizes camera modules to encapsulate constraints and transformations that are enforced while a module is active. A module can be likened to the concept of a shot (an uninterrupted view from a single camera); however, it is possible that a single shot can be a construction of multiple modules. For example, in a single shot the camera may follow a man, then pan to a woman, and then follow the woman. This would require the use of three modules for a single shot. Each module contains an initializer, a local state, a controller, and a constraint list. The initializer, which runs after the activation of a module, sets the camera's initial conditions based on the previous module's state. The local state contains the current camera's position, rotation, field of view, and other parameters. The controller translates a user's input into the camera's states or into its constraints. Finally, the constraint list details what conditions must be held while the module is active. Such constraints may include maintaining a certain camera height or ensuring that certain objects appear in the frame. In addition to the camera modules, the remainder of the system consists of a built-in renderer, an object database for the environment, and a general interpreter, which runs scripts or manages user input. Drucker and Zeltzer do not address what actions are taken when it is impossible for a set of constraints to be resolved.

The cinematography system developed by Bares et. al., called ConstraintCam [26], is also constraint-based, however it has mechanisms for dealing with constraint failures when faced with the complex geometry of a scene. It achieves this by either weakening constraints or through the use of multiple viewports in which a user can see the actions taking place in the environment. It performs this in real-time while faced with an unpredictable environment that can be manipulated by users.

The system architecture contains a narrative planner that instructs the autonomous

characters to perform actions that drive them towards accomplishing their current goal in the 3D world. In addition, a cinematography planner receives the cinematic goals that should be achieved on screen, such as filming a specific character. After receiving a goal, the cinematic planner creates a cinematic constraint problem. These consist of a set of subjects, and for each subject, a set of subject inclusion, vantage angle, shot distance, and occlusion avoidance constraints. If it is unable to find a solution, the solver either attempts to weaken the constraints or decomposes the problem into one that has multiple shots. These multiple shots are then displayed simultaneously. The system does not attempt to take into account the relationship between adjacent shots.

As an alternative to the geometric constraint solving approach to camera placement, the use of film idioms, which are stereotypical ways of filming a series of actions, allows for the creation of more cinematically consistent viewpoints.

Christianson et. al. developed a method for formalizing film idioms into what they call the Declarative Camera Control Language (DCCL) [27]. The primitive concepts in DCCL that combine to make shots and idioms are fragments, views, placements, and movement endpoints. A fragment is similar to a CamDroid module described previously; it "specifies an interval of time during which the camera is in a static position and orientation or is performing a single simple motion". The five fragment types they define are static (camera and actor are static), go-by (actor moves by static camera), panning (camera rotates following moving actor), tracking (camera moves along with moving actor), and point-of-view (camera is positioned at actor's viewpoint). In order to fully specify the fragment, it must be told what actor to follow, over what interval of time, with what camera range (extreme, close-up, medium, full or long), and with what camera placement (external, parallel, internal, or apex). Some types of fragments also need additional information regarding the timing of the movements of the actors and the camera. A shot can consist of one or more fragments, and an idiom can be defined by one or more shots.

The pipeline for their Camera Planning System (CPS) consists of three stages. The three step process begins with the domain-dependent sequence planner, which takes an animation trace as input and outputs the film tree. Next, the DCCL compiler "expands the fragments in each candidate idiom into an array of frame specifications which can be played directly". Finally, the heuristic evaluator "chooses the candidate idiom that renders a scene best, assigning each idiom a real-valued score". The heuristic values smooth transitions and

shot sequences that do not cross the line of action while penalizing extremely short or long shots. Both the DCCL compiler and heuristic evaluator are domain independent.

The Virtual Cinematographer [28], developed by He et. al., also makes use of formalized film idioms. However in this case, the idioms are represented by hierarchical finite state machines and can be executed in real-time interactive environments. In addition, the authors experiment with the cinematographer altering the position of characters controlled by AI or humans in order to achieve better framing. In each time-tick that a frame of animation is generated, the real-time application sends the protagonist-relevant events that are occurring in the form of (subject, verb, object). Then, the Virtual Cinematographer outputs camera specifications and any necessary blocking adjustments to the renderer based on the current events and its previous state.

The Virtual Cinematographer consists of camera modules and idioms. The camera modules, which take in actors as input, describe camera placements based on the actors. Examples of camera modules include apex(actor1, actor2), external(actor1, actor2), and track(actor1, actor2, mindist, maxdist). The camera modules obey the 180-degree rule (a guideline to prevent disruptive cuts), and if two symmetrical camera placements are applicable for a module, the placement is based on which side of the line of action the previous camera module had used. The modules also have the ability to subtly alter the position of the virtual actors in order to improve the appearance of the shot with the given camera placement. The idioms, implemented as a hierarchical finite state machines, instruct the Virtual Cinematographer as to what camera modules to use, when to transition between them, and also when to transition into other idioms. The authors demonstrated this system with an application that simulates a party. Each participant controls a virtual actor in the environment. Through the interface, users can instruct their avatar to perform actions on objects, such as talking to another actor or going to a location.

Amerson and Kime proposed a cinematography system for interactive narratives called FILM (Film Idiom Language and Model) [29], which was later implemented by Jhala [30]. FILM is one part of the Mimesis Project, which is used to tell interactive narratives in gaming environments. Mimesis consists of an AI server that generates plans and sends them to another server to be executed in the game environment. If a user's actions threaten the completion of the plan, then either the player's actions are altered so they are not threatening or the narrative is replanned to adapt to the new environment. The FILM

system, made up of a narrative planner, a Director, a Translator, and a Cinematographer, gives a spectator a cinematic view of the events.

The narrative planner is used to determine what information should be imparted to a viewer for a given scene. Then, a domain dependent Translator reformats the narrative plan operators for the input structure required by the Director, allowing the Director to be domain independent. The Director takes input for the scene and performs a depth-first search on the scene tree. This scene tree consists of a generic shot at its root node, while the three lower levels make distinctions by the number of participants, the emotional effect of the shot, and a keyword list for the finest shot selection. After selecting a scene, the Director passes the information to the Cinematographer. The Cinematographer then finds the optimal placement of the camera, while relaxing constraints if necessary. Jhala later implemented another plan-based cinematography system, Darshak [31], which uses a hierarchical partial order causal link planning algorithm to generate narrative plans.

Chapter 3

Description of Afterthought

3.1 Overview

Afterthought consists of five components - the modified game to be used with the system, the log dispatcher, the narrative pattern matcher, the cinematic discourse generator, and the video renderer. Figure 3.1 illustrates the architecture of the system.

The player initializing the multiplayer match acts as the server for the game. The other players in the match connect to the server as clients prior to the beginning of the game. The log dispatcher also connects to the game server over a socket. As the game is being played, actions performed by the players will trigger log messages that are sent from the game server to the log dispatcher. The log dispatcher then forwards the messages to the narrative pattern matcher and to the cinematic discourse generator. The narrative pattern matcher finds every series of gameplay actions that match the narrative patterns it has been instructed to search for. At the conclusion of the game match, the narrative pattern matcher sends these narratives to the cinematic discourse generator. The cinematic discourse generator then determines which narratives to film and how they should be shot and edited. These instructions for filming are then sent to the video renderer, which records the videos and uploads them to a video sharing web site.

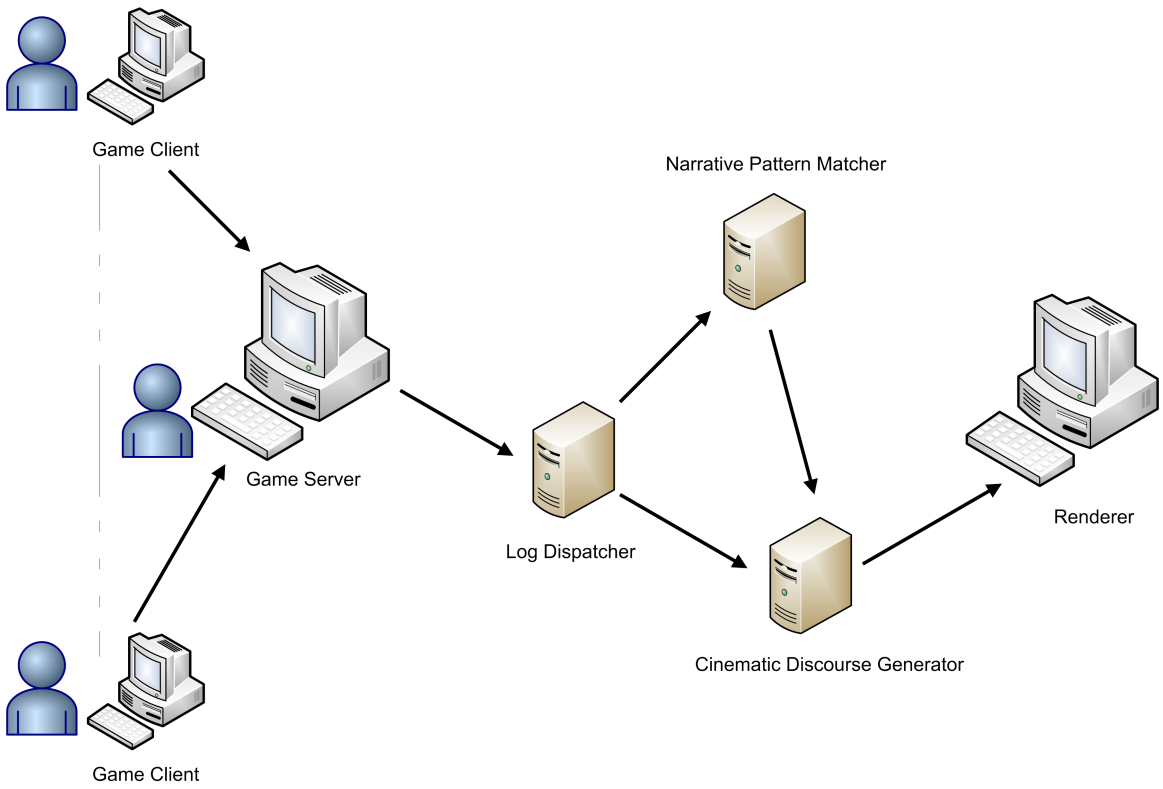


Figure 3.1: Afterthought's architecture.

3.2 Game Requirements

While Afterthought is able to reason about narrative patterns and shot selection independent of the players' game environment, there are a number of requirements that a game must meet for effective integration with the system. First and foremost, the source code of the game must be modifiable. Many games give users the ability to create modifications, or "mods", to the original game. Although typical mods change the appearance of a game or alter the gameplay, the modifications needed to accommodate Afterthought do not need to change the game's standard rules. Instead, changes to the source code are necessary so that the game can send and receive messages from the Afterthought system and to allow the use of custom camerawork during replays. Functionality that must be present in the game, whether included by default or created through a mod, consist of:

- Opening socket connections to outside applications.
- Sending a log message through the socket connection when an important game action occurs.
- Viewing replays of past gameplay sessions.
- Applying slow/fast motion effects during the viewing of a replay.
- Customizable control of the camera's viewpoint during a replay.

For this thesis, a mod that provided for integration with Afterthought was created for *Unreal Tournament 3* (*UT3*), a first person shooter developed by Epic Games. In particular, the examples and evaluation make use of the Capture the Flag game type in which two competing teams attempt to bring the other team's flag back to their home base.

3.3 Game Server and Log Dispatcher

The host of a multiplayer game acts as the game server. When the game session begins, the server establishes a socket connection to the log dispatcher. As important actions take place in the game, the server sends log messages, encoded in XML, out over the connection. See Table 3.1 for every type of action logged by the system for *UT3*'s Capture the Flag game type. See Figure A.2 for an example set of log messages generated from a

gameplay session. The log dispatcher forwards these log messages to the narrative pattern matcher and the cinematic discourse generator. When the game session has completed, the `endgame` log message instructs the narrative pattern matcher that it will receive no more actions and that it can begin sending any matched narratives to the cinematic discourse generator.

Table 3.1: All action types logged in the modification to *UT3*'s Capture the Flag game type.

Message Type
Spawn
Begin Fire
End Fire
Kill
Damage
Switched Weapon
Pickup Item
Flag Score
Flag Pickup
Flag Drop
Flag Return
Sees Player
Stops Seeing Player
Enter Volume
Leave Volume

3.3.1 Unreal Tournament 3 Mod

The section describes the *UT3* mod implementation that was necessary to achieve the recording of the gameplay session and the logging of important actions.

The modification makes use of the demo recording feature available in the Unreal Engine. As mentioned in Chapter 2, a demo file contains the data regarding all of the actions taken by players during a gameplay session. At the start of gameplay, the mod instructs the game to begin a demo recording, and after the match has finished, this recording is saved to a file. This demo file is used later when the game session must be replayed for filming.

In order to log all of the important actions that take place during gameplay, many

objects in the game, such as the weapons, ammunition, powerups, and players' characters, had to be modified. Although their in-game functionality remained the same, the changes allow for the sending of log messages when important actions occur.

3.4 Narrative Pattern Matcher

When the log dispatcher receives messages from the game server, it forwards them to the narrative pattern matcher. The narrative pattern matcher, developed for the Afterthought system by Stephen Roller, finds every series of actions that correspond to predefined narrative patterns. See Figures 3.2, 3.3, and 3.4 for examples of these patterns. An illustration of the internal representation of the pattern shown in Figure 3.4 is depicted in Figure 3.5.

A single action in a pattern is specified by listing the action type followed by the parameters to be bound. For example, `(kill :killerName ?KILLER :victimName ?VICTIM)` matches the kill action with the killer's name and the victim's name bound to variables. `kill` is the type of action, `killerName` and `victimName` are two of the values given with each `kill` log message, and these will be stored in `?KILLER` and `?VICTIM`. The kill message seen in Figure A.2 matches this action with `ATPlayerController_0` binding to `?KILLER` and `ATBot_0` binding to `?VICTIM`.

Users can also append metadata to each action using the `+` operator. The only metadata currently functional is `+focus`, which can be optionally specified to provide the cinematic discourse generator with guidance as to who is of primary importance in the matched action. If no focus is specified, then the default for that action is assumed. For example, by default the `kill` action assumes `killerName` is of primary importance, however using `+focus :victimName` makes the victim the focus of this action.

A series of patterns can be combined to form new patterns as follows:

- `(concat pat1, pat2, ...)` - Defines a pattern in which `pat1` is matched, then `pat2` is matched, etc.
- `(oneormore pat)` - Defines a pattern in which `pat` is matched one or more times.
- `(star pat)` - Defines a pattern in which `pat` is matched zero or more times.

```
# Player A picks up the flag and gets close to scoring but is killed.
(pattern closeattempt
  # The pattern's actions must occur within a span of 90 seconds.
  (within_time 90
    (constrain
      (and (!= ?TEAMA ?TEAMB)
           # The players must be on different teams
           # and the two volumes must be the teams' flag volumes.
           (== ?VOLA (stradd ?TEAMA "FlagVolume"))
           (== ?VOLB (stradd ?TEAMB "FlagVolume")))
      (dontmatch
        :bad_dfa
          (union
            # The player should not score.
            (flagscore :playerPawn ?PAWNA)
            (constrain
              (!= ?NEWCARRIER ?PAWNA)
              # No other player should pick up the flag.
              (flagpickup :flagName ?FLAGA :playerPawn ?NEWCARRIER)
            )
          )
        :good_dfa
        (concat
          # Player A picks up the flag
          (flagpickup :playerPawn ?PAWNA :playerTeamColor ?TEAMA
                      :flagName ?FLAGA)
          # Player A leaves other team's flag volume
          (playerleavesvolume :playerPawn ?PAWNA :volumeName ?VOLB)
          # Player A enters own team's flag volume
          (playerentersvolume :playerPawn ?PAWNA :volumeName ?VOLA)
          # Player B enters Team A's flag volume
          (playerentersvolume :playerPawn ?PAWNB :playerTeamColor ?TEAMB
                              :volumeName ?VOLA)
          # Player B kills Player A
          (kill :killerPawn ?PAWNB :victimPawn ?PAWNA)
          # The flag is returned
          (flagreturning :flagName ?FLAGA)
)))))
```

Figure 3.2: An example pattern for use with the Capture the Flag game type in *UT3* that describes a player's unsuccessful attempt at capturing the other team's flag. In this pattern, a player picks up the opponent's flag, leaves the area near the opponent's base, and enters the area near their own team's base. Then, an opponent enters the area near the base, kills the player, and the flag is returned.

```
# Player picks up the flag, kills zero or more players, then scores.
(pattern capturetheflag
  # The actions must take place within 90 seconds.
  (within_time 90
    (dontmatch
      :bad_dfa
      (union
        # If the flag is returned to its base or if
        # the player drops the flag over the course of
        # these actions taking place, then do not match
        # the pattern.
        (flagreturning :flagName ?FLAGNAME)
        (flagautoreturning :flagName ?FLAGNAME)
        (flagdrop :flagName ?FLAGNAME)
      )
      :good_dfa
      (concat
        # The player picks up the flag.
        (flagpickup :playerPawn ?PAWNA :flagName ?FLAGNAME)
        # The player kills zero or more opponents.
        (star (kill :killerPawn ?PAWNA))
        # The player scores.
        (flagscore :playerPawn ?PAWNA)
      )
    )
  )
)
```

Figure 3.3: An example pattern for use with the Capture the Flag game type in *UT3* that describes a player picking up his opponent's flag, getting zero or more kills, and then scoring.

```
(pattern helpgettingflag
  (within_time 60
    (constrain
      (and (!= ?TEAMA ?TEAMB) (== ?VOLA (stradd ?TEAMA "FlagVolume"))
           (== ?VOLB (stradd ?TEAMB "FlagVolume")))
      (dontmatch
        :bad_dfa
          # Make sure that Player A didn't drop the flag
          (flagdrop :playerPawn ?PAWNA)
        :good_dfa
        (concat
          (dontmatch
            :bad_dfa
              # We don't want the player to leave, then come back
              (playerleavesvolume :playerPawn ?PAWNA :volumeName ?VOLB)
            :good_dfa
              (concat
                # Player A enters the volume near the opposing team's flag
                (playerentersvolume :playerPawn ?PAWNA
                                    :playerTeamColor ?TEAMA :volumeName ?VOLB)
                # Player B returns the flag to their base.
                (flagreturning :playerPawn ?PAWNB :playerTeamColor ?TEAMB)
                # Player B picks up the flag.
                (flagpickup :playerPawn ?PAWNA +focus :flagName)
              )
          )
          # Show all the kills along the way that Player A gets.
          (star(kill :killerPawn ?PAWNA))
          (dontmatch
            :bad_dfa
              # We don't want the player to leave, then come back to score.
              (playerleavesvolume :playerPawn ?PAWNA :volumeName ?VOLA)
            :good_dfa
              (concat
                # Player A enters the volume near his own base.
                (playerentersvolume :playerPawn ?PAWNA :volumeName ?VOLA)
                # Player A scores.
                (flagscore :playerPawn ?PAWNA)
              )))))))
```

Figure 3.4: An example pattern for use with the Capture the Flag game type in *UT3*. Player A goes to get flag, but it's not there. Player B returns it, then Player A picks it up and scores.

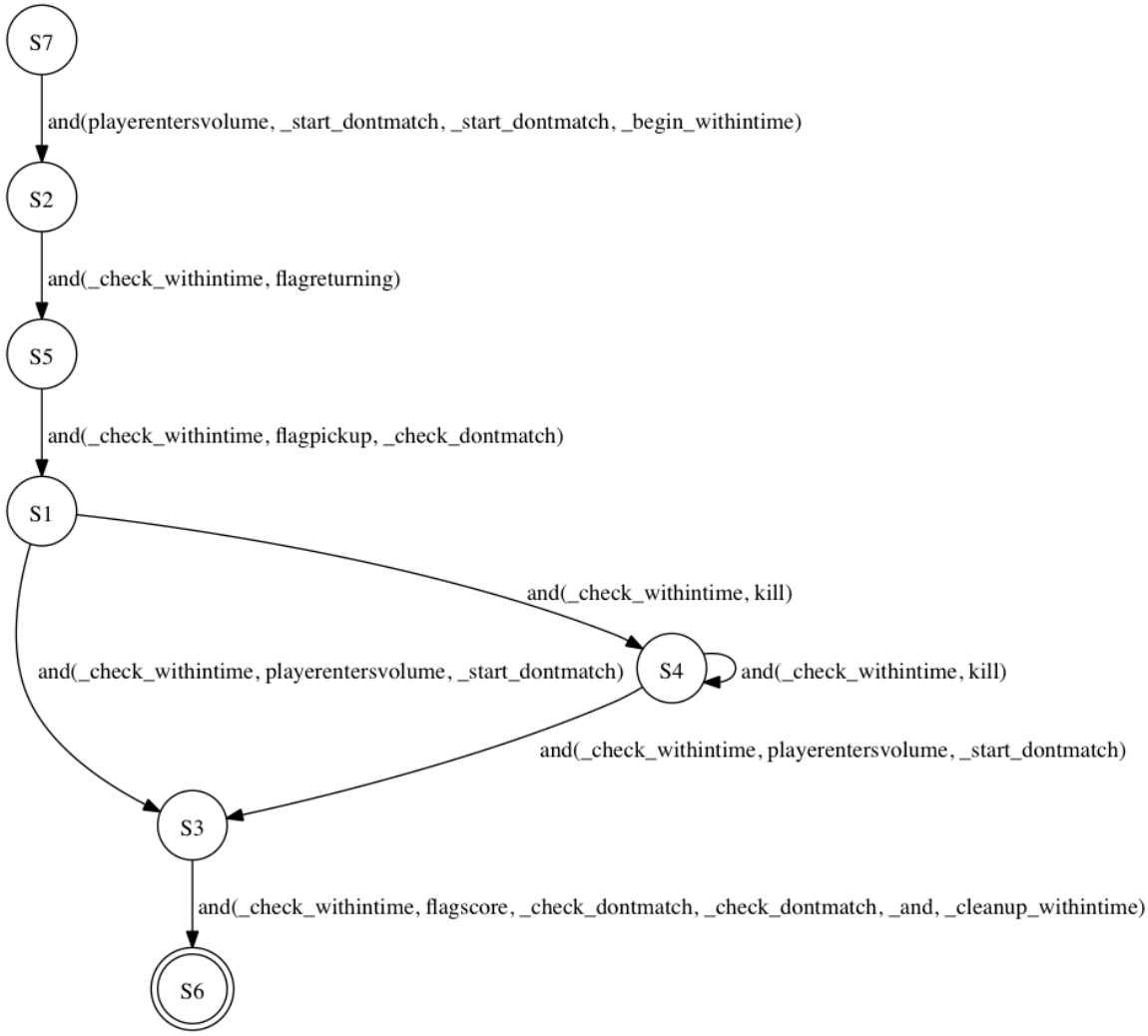


Figure 3.5: A illustration of the internal representation of the narrative pattern specified in Figure 3.4. At each transition, the `withintime` check is performed to see if the specified time has elapsed - if it has, the pattern fails. `dontmatch` ensures that patterns we do not want to match (not shown in illustration) have not occurred between the `dontmatch`'s start and check.

- `(union pat1, pat2, ...)` - Defines a pattern in which `pat1` matches exclusively, or `pat2` matches exclusively, etc.
- `(repeat n pat)` - Defines a pattern equivalent to `(concat pat, pat, pat, pat, ...)`, with `pat` repeating `n` times.

Constraints can be enforced on patterns as follows:

- `(within-time T pat)` - Defines a pattern that machines `pat` only if all the actions in `pat` occur within `T` seconds.
- `(atleast-time T pat)` - Defines a pattern that matches `pat` only if the actions in `pat` occur over at least `T` seconds.
- `(dontmatch :bad_dfa badpat :good_dfa goodpat)` - Defines a pattern that matches `goodpat` only if `badpat` does not match over the course of `goodpat`'s occurrence.
- `(constrain C pat)` - Defines a pattern that matches `pat` if variable constraints `C` are satisfied.

Variable constraints that are available:

- `(== ?A ?B)` - Constraint is satisfied if the values of `?A` and `?B` are equal.
- `(!= ?A ?B)` - Constraint is satisfied if the values of `?A` and `?B` are not equal.
- `(and C1 C2 ...)` - Constraint is satisfied if constraints `C1`, `C2`, etc., are all satisfied.
- `(or C1 C2 ...)` - Constraint is satisfied if any of the constraints `C1`, `C2`, etc., are satisfied.

An additional operator:

- `(stradd ?A ?B)` - Concatenates `?A` and `?B` into a single string value. Useful when specifying variable constraints.

After the narrative pattern matcher receives the `endgame` log message stating that the game match has completed, it sends every set of actions that have matched the defined

patterns to the cinematic discourse generator. Each narrative sent includes its associated pattern name and unique ID number. Also included are the ID number (and optionally the focus) for each ATMessage that comprises the narrative. In addition, the variable bindings used to match the pattern are included, although these are not currently used by the system.

Figure 3.6 shows the output of the narrative pattern matcher given the log messages in Figure A.2 with the patterns defined in Figures 3.2 and 3.3. The pattern in Figure 3.3 matches twice, while the pattern in Figure 3.2 does not match.

```
<Narratives>
    <Narrative name="capturetheflag" ID="1">
        <ATMessageID>17</ATMessageID>
        <ATMessageID>36</ATMessageID>
        <ATMessageID>49</ATMessageID>
        <bindings>
            <PAWNA>ATPawn_2</PAWNA>
            <FLAGNAME>AT_CTFFlag_Blue_0</FLAGNAME>
        </bindings>
    </Narrative>
    <Narrative name="capturetheflag" ID="2">
        <ATMessageID>17</ATMessageID>
        <ATMessageID>49</ATMessageID>
        <bindings>
            <PAWNA>ATPawn_2</PAWNA>
            <FLAGNAME>AT_CTFFlag_Blue_0</FLAGNAME>
        </bindings>
    </Narrative>
</Narratives>
```

Figure 3.6: An example set of narratives sent from the narrative pattern matcher to the cinematic discourse generator. These are the narratives that would be generated if narrative pattern matcher tried to match the pattern in Figure 3.3 with the actions specified in the log messages from Figure A.2.

3.5 Cinematic Discourse Generator

During the gameplay session, the cinematic discourse generator receives the log messages from the log dispatcher. After the gameplay session is complete, it receives all narratives that have occurred from the narrative pattern matcher. After receiving the narratives, it must determine how to film each narrative. Each narrative is comprised of a set of actions that took place during the gameplay session. These actions may have occurred very close or very far apart in time. Each action may feature the same players, or perhaps a wide variety of players. The problem of how to create an effective video highlighting these actions is addressed here.

The running example in this section will follow from the log messages in Figure A.2, sent from the log dispatcher, and the narratives in Figure 3.6, sent from the narrative pattern matcher. In particular, it will focus on the narrative with `ID` = 1.

For a particular narrative, the output of the cinematic discourse generator is a *camera event*. The camera event contains the full specifications that the renderer and game engine need to film the narrative. Figure 3.7 shows the major steps taken to generate a camera event.

```
1. Load the camera action series for each action
   that occurs in the narrative.
2. Resolve the timing conflict between all camera actions.
3. Add the slow motion effects.
4. Add the recording actions.
5. Load each camera action's camera modifiers.
```

Figure 3.7: The steps taken to generate a camera event.

To begin the construction of a camera event, the first step is to load the *camera action series* that is associated with each action in the narrative. A camera action series defines the optimal way of filming an action. An action is not guaranteed to be filmed as described in its camera action series because in the generation of the camerawork for an entire narrative, certain elements may have to be sacrificed in order to effectively create a video. The narrative used for the running example features a flag pickup, a kill, and a flag

score. The camera action series for each of these actions are shown in Figures A.3, A.4, and A.5. As seen in those figures, each *camera action* in a camera action series is defined with a file name, its timestamp and actor variables, and its constraints.

The file name points to details about the implementation for a specific camera action and will be discussed in Section 3.5.4. The timestamp variables include the time that the camera action starts and the time that the camera action ends. In addition, a transition time, which will be discussed in Section 3.5.1, may be specified. The position and rotation of the camera is usually defined relative to the actors specified in the actor variables. In the constraints section, each camera action has an importance rating. An importance rating of 10 indicates that the camera action is a *primary camera action*, meaning that it must be included in the final camera event that is generated. Each primary camera action also has a must-include timestamp in the constraints, which is a timestamp during which this camera action must be active. Typically, this is the timestamp at which the associated action has occurred. Any camera action with an importance rating less than 10 is considered a *secondary camera action* and may be completely removed in the final camera event that is generated.

Notice the use of square brackets in the camera action series. These indicate variables that will be replaced using the camera action series' associated log message. For instance, the first action in the example narrative points to the log message with `ID` = 17, a `flagpickup` action. In the camera action series for this action, `[timestamp]` is replaced with `23.5101`, `[playerPawn]` is replaced with `ATPawn_2`, and `[flagName]` is replaced with `AT_CTFFlag_Blue_0`.

3.5.1 Resolving Timing Conflicts Between Camera Actions

After loading each camera action series for the running example, the camera event looks like that shown in Figure 3.8.

As only one camera action may be active at once, conflicts exist among some of the starting and ending times for the camera actions, as shown in Figure 3.9.

These conflicts need to be resolved before a video can be created. First, the timing conflicts regarding the primary camera actions are considered. See Figure 3.10 for the algorithm used to resolve these conflicts.

The first step in resolving conflicts between primary camera actions is to check

Secondary Camera Action for Flag Pickup Start: 14.5101 End: 17.5101	Secondary Camera Action for Flag Pickup Start: 17.5101 End: 20.5101	Primary Camera Action for Flag Pickup Start: 20.5101 End: 26.5101	Secondary Camera Action for Kill Start: 38.4369 End: 41.4369	Primary Camera Action for Kill Start: 41.4369 End: 47.4369	Secondary Camera Action for Kill Start: 47.4369 End: 50.4369	Secondary Camera Action for Flag Score Start: 44.6511 End: 47.6511	Secondary Camera Action for Flag Score Start: 47.6511 End: 50.6511	Primary Camera Action for Flag Score Start: 50.6511 End: 56.6511
0	1	2	3	4	5	6	7	8

Figure 3.8: The example camera event after loading the camera action series for each action in its associated narrative. Each box represents a single camera action and is labeled with text that indicates its status as a primary or secondary camera action, its associated type of action from the narrative, and its starting and ending time. The smaller boxes below each camera action represent array indices. Time proceeds from left to right, although timing conflicts exist in this example, as shown in Figure 3.9.

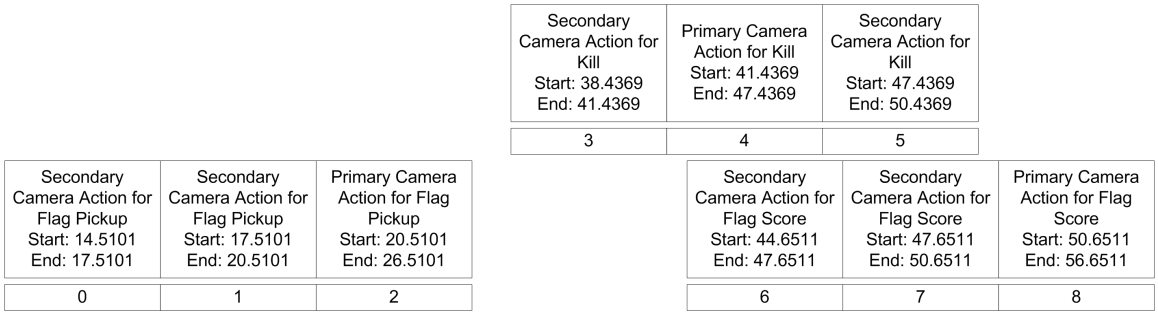


Figure 3.9: Illustrating how the camera actions shown in Figure 3.8 overlap with each other. The overlap indicates that there are timing conflicts that need to be resolved (boxes not to scale of timeline).

```
For each primary camera action:
        If a spawn time for an actor is later then start time:
            Adjust starting time to latest actor spawn time.

For each primary camera action:
        If the camera action overlaps with the next primary camera action:
            Split the difference between the must-include timestamps to
            compromise, but don't extend the camera action beyond
            its existing ending time.
            If the two camera actions have the same primary target:
                Set the transition time on the next primary camera action
                to its must-include timestamp.
```

Figure 3.10: The algorithm used to resolve timing conflicts between primary camera actions.

the spawn times for the actors associated with the camera actions. The spawn time refers to the time that an actor is instantiated in the game world. As the camera's position and rotation will be relative to their actors, a camera action should not begin until all of its associated actors have spawned.

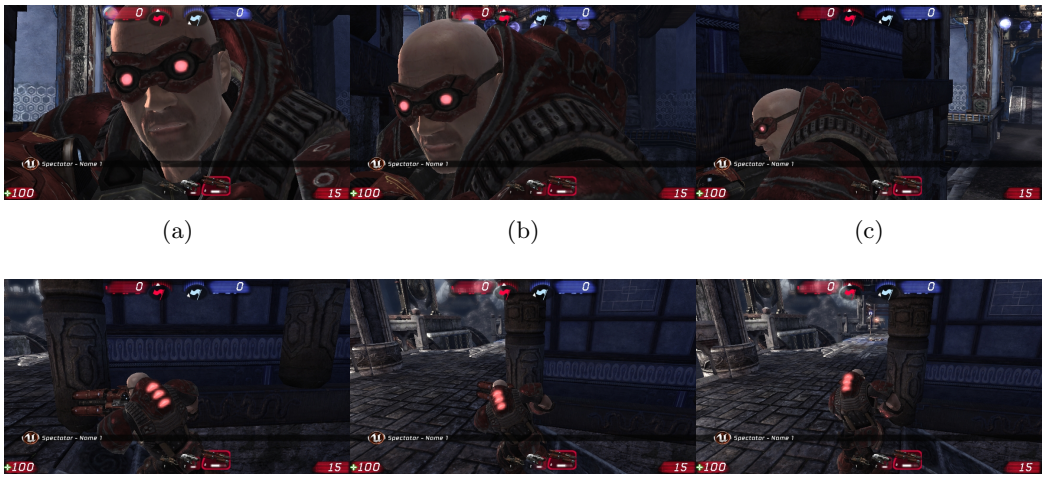
Next, each primary camera action is checked to see if it overlaps with the next primary camera action in the camera event. If it does, then a compromise needs to be made to ensure that only one camera action will be active at a time. To solve this, we check the must-include timestamps for the camera actions and set the ending time of the camera action (and starting time of the next camera action) to `(must-include timestamp for camera action) + (must-include for next camera action - must-include timestamp for camera action) / 2`. However, the first camera action is not allowed to extend beyond the ending point that had already been set for it. Next, if the two camera actions have the same *primary target*, then we allow a *transition* to occur between the two camera actions.

A primary target refers to the actor for which the camera's position is relative. A transition refers to interpolating the position, rotation, and field of view from one camera action to another. This is opposed to cutting, which will immediately change these camera values. Using transitions as opposed to cuts when two consecutive camera actions have the same primary target allows for more dynamic camerawork and also aids in the prevention of cuts that would be disruptive to the viewer. See Figure 3.11 for screenshots taken during a transition. Transitions are accomplished in the game engine through the use of spherical coordinates, with the primary target's location considered the center, so camera actions with different primary targets are not considered for transitions. If camera actions with different primary targets were considered for transitions, then there would need to be some reasoning about the level geometry or some implementation of pathfinding used during the recording of the video in order to effectively relocate the camera from one target to another.

In our example camera event, no conflicts exist between the primary camera actions, so no changes will need to be made to it.

Next, conflicts affecting the secondary camera actions are considered. As opposed to primary camera actions, secondary camera actions will not have any of their starting or ending times adjusted to resolve conflicts. Instead, if a conflict exists, they are completely removed. The reasoning for this is discussed in Section 3.5.2.

The first check for conflicts with secondary actions is shown in Figure 3.12. Simply,



(a) (b) (c)

(d) (e) (f)

Figure 3.11: Screenshots illustrating a transition. The first screenshot is a close-up, while the final is a long shot behind the actor. The frames in between show the transition taking place from one to the other.

if any of the actors associated with the camera action spawn between the starting time and ending time of the camera action, then the camera action is removed from the camera event. After removing a secondary camera action, we need to check if the next camera action in the camera event has a transition from it. Since the camera action will no longer exist, any transitions from it are invalid. As primary camera actions may have already had transitions put in place to resolve their conflicts, we need to make sure not to supersede them.

```
For each secondary camera action:
        If any of the targets spawn during the camera action:
            Remove this camera action.
            If the next camera action is not a primary
            camera action with a transition from another
            primary camera action:
                Remove the transition time from the next camera action.
```

Figure 3.12: The algorithm used to resolve timing conflicts regarding the spawn times of actors for secondary camera actions.

No actor spawn times conflict with the secondary camera actions in the running example, so no changes are made in this step.

Finally, we consider conflicts between secondary camera actions and other camera actions, as shown in Figure 3.13.

When attempting to resolve these conflicts, camera actions in the camera event are looped through in reverse order. This allows us to give preference to secondary camera actions that occur before the primary camera action in its series. If the camera action at the previous index has an ending time that is greater than the current camera action's start time, then there is a conflict. If the camera action at the previous index is a primary camera action, then it has preference over secondary camera actions, so the current camera action is removed. In addition, the camera action at the index past the current camera action has its transition time removed, if it is not a primary camera action with a transition from another primary camera action. If the camera action at the previous index is not a primary camera action, then we remove that camera action and remove the transition time for the current camera action. If we have removed any camera action, then we check the same index again next iteration.

```
For each camera action in the camera event, from index i = n-1 to 0:
    If the camera action at i is a secondary camera action:
        If i-1's ending time is greater than i's start time:
            If i-1 is a primary camera action:
                Remove camera action at i.
                If camera action at i+1 is not a primary camera action
                with a transition from another primary camera action:
                    Remove i+1's transition time.
            Else:
                Remove camera action at i-1.
                Remove the transition time for camera action at i.
            Check camera action at index i again next iteration.
```

Figure 3.13: The algorithm used to resolve timing conflicts regarding secondary camera actions. N = number of camera actions currently in the camera event.

In the running example, the camera actions in indices 5 and 6, as seen in Figure

3.8, are now removed due to conflicts.

After this process is completed, the timing conflicts for all camera actions in a camera event have been resolved.

3.5.2 Adding Slow Motion Effects

The next step is to add a slow motion effect to the primary camera actions. Adding slow motion to important shots allows the viewer more time to comprehend what is occurring and intensifies the action taking place [32]. This is especially useful since in the trimming of primary camera actions to avoid timing conflicts, their duration may be very short. While all primary camera actions have some slow motion effect applied, any primary camera action whose duration is shorter than a certain threshold will have a more pronounced slow-motion effect. The amount of slow motion applied increases linearly from the start of the camera action until its must-include timestamp, then decreases linearly to the end of the camera action. The slow motion effect is not applied to secondary camera actions because this would place undue emphasis on these supplementary shots.

3.5.3 Adding Recording Actions

The final step in creating a camera event is to add the actions that are used to instruct the renderer when to start and stop capturing the game's viewport to a video file, as well as when the game should fast forward past gameplay that will not be filmed.

The following guidelines are followed:

- Before the first camera action in the camera event, a start capturing action is added, with a starting time equal to the camera action's starting time.
- Any time there is a gap between the ending time of a camera action and the starting time of the next camera action, a stop capturing action is added, with a starting time equal to the first camera action's ending time, followed by a start capturing action, with a starting time equal to the second camera action's starting time.
- Before every start capturing action, a fast forwarding action is added. The starting time is set at its preceding stop capturing action, or to 0.0 if it is being inserted before the first start capturing action in the camera event. The ending time, indicating when

the fast forwarding should cease, is set at the starting time of the start capturing action. The fast forwarding action is used to skip past portions of gameplay that will not be included in the final recording, since depending on the game engine it may not be possible to jump directly to a particular time. The use of fast forwarding greatly decreases the amount of time needed to generate a video.

- Finally, a compile video action is added at the conclusion of the camera event, with a starting time equivalent to the starting time of the final stop capturing action. This action is used to instruct the renderer that all filming is complete for this narrative.

3.5.4 Loading Camera Modifiers

As shown in Figures A.3, A.4, and A.5, each camera action has an associated file name. This file, an example of which is seen in Figure 3.14, contains the timestamps and the *camera modifiers* that are used with that camera action. A camera modifier consists of specifications for one or more of a camera placement's degrees of freedom. These degrees of freedom include position (x, y, and z), rotation (yaw, pitch roll), and field of view. The camera modifiers used for the camera action file in Figure 3.14 are show in Figures 3.15, 3.16, and 3.17. Angle values are specified in degrees. The units for distances, however, are particular to a game engine, therefore a specific set of camera modifiers needs to be developed for each game engine used with Afterthought.

The camera event shown in Figure A.6 was generated by using the camera action series show in Figure A.3, A.4, and A.5 with the narrative with `ID` = 1 in Figure 3.6 and the log messages from A.2.

3.6 The Renderer

The XML camera specifications for recording all the narratives are sent from the cinematic discourse generator to the renderer, which is the component responsible for rendering the videos of the narratives. The renderer instructs the game to begin playback of the gameplay session. As soon as the game begins playback, it requests the camera event from the renderer. After receiving it, the game, as instructed by the first step in the camera event, fast forwards to the timestamp where the first start-capture action needs to be executed.

```
<camera-action>
     <timestamps>
          <start-time>[start-time]</start-time>
          <end-time>[end-time]</end-time>
          <transition-end-time>[transition-end-time]</transition-end-time>
     </timestamps>
     <camera-modifiers>
          <camera-modifier>
               <file-name>CamMod-LocationRelative-BackLong</file-name>
               <start-time>[start-time]</start-time>
               <end-time>[end-time]</end-time>
               <target>[target]</target>
          </camera-modifier>
          <camera-modifier>
               <file-name>CamMod-RotationRelative-Forward</file-name>
               <start-time>[start-time]</start-time>
               <end-time>[end-time]</end-time>
               <target>[target]</target>
          </camera-modifier>
          <camera-modifier>
               <file-name>CamMod-FOV-Normal</file-name>
               <start-time>[start-time]</start-time>
               <end-time>[end-time]</end-time>
          </camera-modifier>
     </camera-modifiers>
</camera-action>
```

Figure 3.14: An example camera action for a long shot following behind the player. The camera modifiers used will locate the camera in back of the target at a long distance, and will keep its rotation facing the same direction as the target. In addition, it will use a normal field of view.

```
<camera-modifier>
    <type>ATCamMod_LocationRelative</type>

    <start-time>?</start-time>
    <end-time>?</end-time>
    <target>?</target>

    <direction-yaw>180</direction-yaw>
    <direction-pitch>0</direction-pitch>
    <direction-roll>0</direction-roll>
    <distance>100</distance>
</camera-modifier>
```

Figure 3.15: CamMod-LocationRelative-BackLong: A camera modifier that sets the location of the camera based on a vector offset relative to a specified target actor. In this case, the camera will be placed directly behind the target actor at distance of 100 units. The start time, end time, and target will be specified when the camera modifier is instantiated. The direction angles are specified in degrees, and the distance is specified units particular to the Unreal Engine.

```
<camera-modifier>
    <type>ATCamMod_RotationRelative</type>

    <start-time>?</start-time>
    <end-time>?</end-time>
    <target>?</target>

    <yaw>0</yaw>
    <pitch>0</pitch>
    <roll>0</roll>
</camera-modifier>
```

Figure 3.16: CamMod-RotationRelative-Forward: A camera modifier that sets the rotation of the camera to be equal to that of its target actor. The start time, end time, and target actor will be specified when the camera modifier is instantiated. The direction angles are specified in degrees, and the distance is specified in units particular to the Unreal Engine.

```
<camera-modifier>
    <type>ATCamMod_FOV</type>

    <start-time>?</start-time>
    <end-time>?</end-time>

    <FOV>90</FOV>
</camera-modifier>
```

Figure 3.17: CamMod-FOV-Normal: A camera modifier that sets the field of view to be its normal value. The start time and end time will be specified when the camera modifier is instantiated. The FOV angle is specified in degrees.

When a start-capture action executes, a message is sent from the game to the renderer telling it to begin capturing the game's viewport to a video file. The renderer utilizes Fraps [33], a video capture utility, to accomplish this. The subsequent camera actions are applied during playback so that the appropriate camera shots are used to capture the actions that take place in the game. When a stop-capture action is executed, a message is sent from the game to the renderer telling it to stop capturing the game's viewport to a video file. The game then fast forwards again to the next start-capture to begin capturing the next sequence of actions in the narrative, and so on. The final action executed will be the compile-video action, which sends a message to the renderer that all video files have been captured for this narrative and it should now combine them into a single file.

This process begins by creating a text file containing an AviSynth [34] script that consists of commands to concatenate all the video clips depicting the narrative. See Figure 3.19 for an example script generated by the renderer. The beginning of the final video will fade in from black, and it will fade out to black at its conclusion. Each recorded sequence transitions to the next using a dissolve. A dissolve consists of fading out from one clip while at the same time fading into the next. Filmmakers use the dissolve transition when they need to "form a bridge between disparate times and places"[5]. In addition, it has the effect of "soften[ing] an otherwise terrible cut" [35]. Given both of these properties, it serves as the ideal method for transitioning between two clips that are are separated by a gap in time

(a) Flag pickup - front long shot.

(b) Flag pickup - close up shot.

(c) Flag pickup - in line with two targets.

(d) Kill - front long shot.

(e) Kill - in line with two targets

(f) Flag score - close up shot.

(g) Flag score - in line with two targets.

Figure 3.18: Screenshots from the video generated using the camera event from Figure A.6 with the game session described in the log messages in Figure A.2.

```
# Play the first clip followed by the second clip,
# with a 30 frame dissolve transition between the two.
Dissolve(Avisource("sequence-1.avi"), Avisource("sequence-2.avi"), 30)

# Play the next clip, with a 30 frame dissolve from the previous clip.
Dissolve(last, Avisource("sequence-3.avi"), 30)

# Fade in over 15 frames at the start of the video.
FadeIn(15)

# Fade out over 15 frames at the end of the video
FadeOut(15)
```

Figure 3.19: An example AviSynth script generated by the renderer to concatenate the videos comprising a narrative.

and that may also cut together harshly.

After the AviSynth script is generated, the file is opened by VirtualDub [36], a video processing utility. VirtualDub executes the instructions to concatenate the clips and also compresses the movie to a smaller file size. After the compression is complete, a utility based on an open source library [37] uploads the generated video to a YouTube account. After the completing the upload, the process begins for the next narrative and continues until finishing the recording of all narratives.

Figure 3.18 features screenshots from the final video generated by from the running example.

3.6.1 Unreal Tournament 3 Mod

The section discusses the details specific to the *UT3* mod implementation that was necessary to achieve the playback of the gameplay session, custom camera control, and video capture.

In order to film the replays, Afterthought utilizes the demo recording feature available in Unreal Engine. During gameplay, Unreal makes a record of all actions that players perform in the game. After a match has finished, this record is saved to a file. This

file can then be loaded in order to watch a replay of the match. Typically when watching a replay, users can either focus their viewpoint on a single player or enable the "free camera" mode in which they have manual control of the camera and can float freely around the game world. In the Afterthought system, the code that determines the viewpoint of the player has been modified so that we are able to record using our custom camerawork.

After receiving the XML camera specifications, the Renderer copies the demo file generated on the game server to a local folder designated by the Unreal Engine to contain demo files for playback. Next, the renderer must instruct the Unreal Engine to playback this demo. To accomplish this, it simulates the keystrokes necessary to open a command line console in the game, type the command to play back our demo file, and execute the command.

Chapter 4

Preliminary Evaluation

This chapter discusses the preliminary evaluation conducted to assess the capabilities of some components of the Afterthought system.

In this evaluation, we sought to determine the effectiveness of Afterthought providing dynamic camerawork during a replay, in contrast to the standard third-person view used by many games. To do this, human subjects played short rounds of the Capture the Flag game type in *Unreal Tournament 3* while making use of the Afterthought system. Afterthought generated cinematics based on the subjects' gameplay, both using our custom camerawork and the default third-person view. Subjects were then presented with the videos to review. Subjects rated the videos on a number of qualities and indicated their level of participation in the featured events. Details of the experiment and a discussion of our results are provided below.

4.1 Participants, Materials, and Apparatus

The participants consisted of 18 graduate and undergraduate students from North Carolina State University who took part in groups across three study sessions.

Each subject received a two reference sheets, shown in Figures B.1 and B.2, describing the keyboard and mouse controls needed to play as well as the rules for the Capture the Flag game type in *UT3*. Additional paper materials, two surveys shown in Figures B.3 and B.4, are described in Section 4.2.

The computers used were Dell desktops, each with an Intel Core 2 2.40 GHz

processor, 2 GB of RAM, and an NVIDIA GeForce 7900 GS graphics card. Participants used headphones to hear sounds from the game. The evaluation utilized version 2.1 of *UT3*. Each Capture the Flag match was played on the map titled "Reflection", which comes included with *UT3*. Modifications to the map and game type enabled our logging capabilities, however they did not affect gameplay. For the matches, the translocator item, which gives players the ability to teleport around the map, was disabled. Each game match lasted approximately 10 minutes.

4.2 Procedure

At the beginning of a study session, each subject was assigned to a computer in a computer lab. They took few minutes to review the reference sheets before playing a 5 minute practice Capture the Flag match. The purpose of the practice match was to allow participants to gain familiarity with the game. After the conclusion of this practice session, the participants took part in a Capture the Flag match during which the Afterthought system logged their actions. At the completion of this match, the participants then watched a replay of the game from the same first-person perspective they used while playing. As they watched the replay, they completed the survey shown in Figure B.3, which includes a question asking them to choose three significant gameplay moments. After all the participants completed this survey, they then watched four videos generated by Afterthought and completed a survey, shown in Figure B.4, for each. This survey asked them to rate each video on a number of qualities (Exciting, Humorous, Dramatic, Interesting, Coherent) on a 7 point Likert scale. They also responded as to whether they witnessed or participated in the depicted events and whether any of the events were described in their answers to the first question on the previous survey. In addition, they provided their general thoughts regarding the video.

4.2.1 Deciding What Narratives To Film

A large number of narratives may be sent from the narrative pattern matcher to the cinematic discourse generator, as it simply sends each and every narrative that is found over the course of a game session. However, it would be prohibitive to record a video for each narrative for our evaluation, so a method was devised to determine which narratives should

have their videos rendered. First, each type of narrative was given a ranking - the more desirable the type of narrative is to film, the higher rank it received. The two narratives that occurred during a game session with the highest narrative type ranking (of different narrative types, ties broken by the number of camera actions in the narrative's generated camera event) were chosen to be filmed.

Also generated were two videos that did not take advantage of several aspects of the system. These two videos filmed the same narratives as the previously mentioned videos. However, they did not have any slow-motion effects applied and they were shot entirely from the third-person camera view used in a typical video game replay. They were edited the same way as the previous two videos.

To negate the effects of watching the videos in a certain order, each subject in a session received a unique order in which to view them, as explained in Table 4.1. The viewing order chosen was assigned randomly based on the time their game client connected to the game server.

Table 4.1: The possible viewing orders assigned to the subjects. Narrative 1 and Narrative 2 were filmed both with our custom camerawork (A) and without (B).

	Viewed First	Viewed Second	Viewed Third	Viewed Fourth
Ordering 1	1-A	1-B	2-A	2-B
Ordering 2	1-A	1-B	2-B	2-A
Ordering 3	1-B	1-A	2-A	2-B
Ordering 4	1-B	1-A	2-B	2-A
Ordering 5	2-A	2-B	1-A	1-B
Ordering 6	2-B	2-A	1-A	1-B
Ordering 7	2-A	2-B	1-B	1-A
Ordering 8	2-B	2-A	1-B	1-A

4.3 Results

Due to technical difficulties, seven participants were not able to view the videos generated by Afterthought immediately following their viewing of the full replay. The responses for three of these participants were subsequently discarded for incomplete responses

on their surveys. Results are presented both including and excluding the remaining four participants, who watched the videos and completed the surveys remotely.

In the following tables, Type A indicates the set of generated videos using slow-motion effects and custom camerawork, while Type B indicates the set of generated videos that did not use slow-motion effects and only featured the default third-person camerawork. Each set contains both the 30 participant/narrative pairings, which includes all participants, and the 22 participant/narrative pairings, which removes the previously mentioned four participants.

The null hypothesis for this experiment was that the mean scores of the videos of Type A will be less than or equal to the mean scores of the videos of Type B in the five qualities being examined. The alternative hypothesis stated that the mean scores of the videos of Type A will be greater than the mean scores of the videos of Type B in the five qualities being examined. To find support for the alternative hypothesis, 1-sided t-tests were performed with a confidence interval of 0.01 (Bonferroni correction, 0.05/5). The reason for choosing t-tests was because the true standard deviation was not known, and the number of sample survey responses was small (less than 40), which is a scenario that is appropriate for a t-distribution of data.

Table 4.2: The mean scores given to the specified qualities in each group.

Group	Exciting	Humorous	Dramatic	Interesting	Coherent
30 pairings, Type A	4.97	2.57	4.87	4.83	4.77
30 pairings, Type B	4.70	1.97	4.00	4.77	4.80
22 pairings, Type A	5.00	2.27	5.05	4.77	4.59
22 pairings, Type B	4.63	1.95	4.27	4.77	4.86

Table 4.3: The mean of the difference between the scores given by each participant for the video of Type A and the video of Type B of the same narrative.

Group	Exciting	Humorous	Dramatic	Interesting	Coherent
30 pairings	0.27	0.60	0.87	0.07	-0.03
22 pairings	0.36	0.32	0.77	0.00	-0.27

Table 4.4: The standard deviation of the difference between the scores given by each participant for the video of Type A and the video of Type B of the same narrative.

Group	Exciting	Humorous	Dramatic	Interesting	Coherent
30 pairings	1.57	1.48	1.55	1.55	1.47
22 pairings	1.59	1.32	1.51	1.57	1.55

Table 4.5: T-statistic for the difference between the scores given by each participant for the video of Type A and the video of Type B of the same narrative.

Group	Exciting	Humorous	Dramatic	Interesting	Coherent
30 pairings	0.93	2.23	3.07	0.24	-0.12
22 pairings	1.07	1.13	2.40	0.00	-0.83

In the 30 pairings group, with 29 degrees of freedom, a critical value of 2.462 was used, while for the 22 pairings group, with 21 degrees of freedom, a critical value of 2.518 was used. These critical values were taken from a t-distribution table of critical values. Using these values with the data shown in Table 4.5, the null hypothesis was rejected in favor of the alternative hypothesis in the 30 pairings group in regards to the Dramatic quality.

Another item of note are the results of the questions asking whether or not the subject witnessed/participated in the events depicted in the video and if the subject had selected any of the moments in the video as significant moments on their Gameplay Survey. For these results, the four participants discussed earlier were not included.

Table 4.6: Responses to the question, "When you were playing the game, did you witness or participate in any of the depicted events in the video?"

Total Responses	All	Some	None
22	4	11	7

Table 4.7: Responses to the question, "Did you describe part or all of the events in this video in your gameplay survey?"

Total Responses	Yes	No or N/A
22	9	13

4.4 Discussion

The data supports that the participants found the videos of Type A more dramatic than the videos of Type B. It appears that the dynamic camerawork and slow-motion effects successfully added more tension to the generated highlight videos.

As shown in Tables 4.6 and 4.7, Afterthought was able to find gameplay moments that some players had not witnessed or that they did not initially choose as moments on their Gameplay Survey. It should be noted that some participants responded as All or Some for Table 4.6's question and Yes for Table 4.7's question based on their activity in the background of shots and not on the actions that Afterthought had matched.

A number of insights were gained from the participants' answers to the short answer questions on the video surveys.

Some participants expressed their appreciation of the application of slow-motion to emphasize important gameplay moments, however some felt that at times it was overused or misused. This demonstrates that perhaps the slow-motion effects should only be applied to the most important actions, or only certain types of actions, rather than for every action that occurs in the matched pattern.

One aspect of the videos that received multiple negative remarks was the use of close-up shots. In film, a close-up shot, which is a shot that is tightly framed on a subject's face, "provides the greatest psychological identification with a character" [35]. In addition, a transition from a medium or long shot of a subject to a close up can add tension to a scene [32]. However, these benefits did not appear to translate to their use in the videos generated by Afterthought. One primary difference may be that in a live-action film, an actor's face can reveal emotional cues that are effectively conveyed through a close-up, while in the game a character's model remains static throughout. Also, some participants remarked that they wanted to see the action taking place around the character rather than see a close-up shot.

The dislike of close-ups may also be related to another unfavorable aspect of the videos. During the replay of a game session's demo file, the character animations sometimes exhibit jerky motions, and this effect is exacerbated during a character's close up shot. Due to the closed source of critical components of *UT3*'s game engine, it was not possible to make an attempt to remedy this shortcoming.

Another feature that drew mixed reactions was the the use of dissolves as transitions between shots. Some expressed gratitude for skipping past unnecessary sections of gameplay. However, some expressed dislike for their use when only a small amount of time was removed by use of the dissolve. Particularly, those whose character was the primary focus during a transition noticed these occurrences and remarked that it gave the wrong impression. This indicates that it may be useful to allow filming during uninteresting moments if the gap in time is short enough in duration.

In one particular video, two important actions occurred near simultaneously and Afterthought did not effectively present them, as participants made comments regarding confusion at the choice of shot selection for this moment. Ideas for addressing this issue of simultaneously occurring actions are presented in Chapter 5.

Chapter 5

Conclusions and Future Work

This thesis presented Afterthought, a system that identifies complex interactions between players in 3D games and generates videos showcasing them.

It accomplishes this through the use of a modified game, a log dispatcher, a narrative pattern matcher, a cinematic discourse generator, and a video renderer. The actions of players in the game are able to be captured using the modified game and the log dispatcher. The narrative pattern matcher can find series of actions that correspond to predefined narratives. The cinematic discourse generator is able to determine how to shoot and edit these narratives. Finally, the renderer can record the videos and upload them to a web site so they can be easily viewed by all of the game's participants.

A preliminary evaluation was performed to determine the effectiveness of Afterthought's capabilities of providing dynamic camerawork during a replay. The data gathered supported the claims that the custom camerawork provided for more humorous and more dramatic cinematics in comparison to the standard third-person view. Responses from subjects provided included both positive remarks and constructive criticism indicating where improvements need to be made.

5.1 Future Work

There are numerous opportunities for future work to build on top the work presented in this thesis.

Given the distributed structure of Afterthought, the narrative pattern matcher

can easily be replaced with different log analyzers. These may use different techniques to discover narratives or may instead take a different approach. For example, perhaps a log analyzer could be developed to detect patterns of good or bad behavior by players, thereby using the system as an after-action review tool. Another useful addition to the pattern matcher could be the ability to constrain narratives by the current state of the world, rather than specific actions that are occurring.

Figure 5.1: A screenshot from the TV series *24* illustrating the use of multiple viewpoints, which allows the audience to witness simultaneously occurring actions that take place in different locations.

One problem to be addressed in future work is improving the system's ability to capture narratively significant actions that are temporally close but spatially distant. Currently, the solution to address this problem is to alter the speed of the camera shots used to capture them. This allows the actions to be presented in such a way that viewers can comprehend what is occurring without rapid cuts. However, this approach may not be appropriate for every gaming scenario. In addition, it does not solve the problem of filming many actions that occur at nearly the exact same time in different locations. An alternative solution may be to show the audience multiple viewpoints simultaneously. This was implemented successfully by Bares in ConstraintCam [26] and additional work may allow for this to be integrated into Afterthought. AviSynth provides a method, through its "Overlay" filter, to superimpose one video on top of another. Using this technique, the renderer would need to recognize when multiple passes through the replay are necessary to capture certain actions, and then be able to combine the videos appropriately. This method of presenting multiple viewpoints simultaneously has been successfully used in the media, such the television series *24* [38] as illustrated in Figure 5.1, and could be an effective

addition to the system.

Another improvement could be the addition of more camera action series for each gameplay action. Then, the cinematic discourse generator could weigh the advantage or disadvantage of using certain combinations.

Currently, the occlusion detection employed in the *Unreal Tournament 3* modification used by Afterthought is very basic. If the viewpoint to the primary target of the shot is occluded by level geometry, then the camera is simply placed in front of the occlusion. This can occasionally result in abrupt changes in the viewpoint. A more robust detection strategy could employ prediction techniques to proactively adjust the viewpoint in an attempt to avoid any potentially disruptive camera movements.

Other improvements are inspired by the feedback from our preliminary evaluation, including a more selective application of both the slow-motion effect and of dissolves as transitions.

Finally, another goal is to make use of the Afterthought system with more games. Currently modifications have only been developed for *Unreal Tournament 3*, however other modifiable or open source games may also be compatible with the appropriate alterations.

Bibliography

[1] *Madden NFL 2010*. EA Sports, 2009.

[2] *Halo 3*. Microsoft Game Studios, 2007.

[3] *Mario Strikers Charged*. Nintendo, 2007.

[4] Daniel Arijon. *Grammar of the Film Language*. Focal Press Limited, 1976.

[5] Stephen D. Katz. *Film Directing Shot by Shot: Visualizing from Concept to Screen*. Michael Wiese Productions, 1991.

[6] Matt Kelland, Dave Morris, and Leo Hartas. *Machinima: Making Animated Movies in 3D Virtual Environments*. Thompson Course Technology PTR, 2005.

[7] *Unreal Tournament 3*. Epic Games, 2007.

[8] *Call of Duty: Modern Warfare 2*. Infinity Ward, 2009.

[9] *Call of Duty*. Activision, 2003.

[10] Yun-Gyung Cheong, Arnav Jhala, Byung-Chull Bae, and R. Michael Young. Automatically generating summary visualizations from game logs. In *Conference on Association for the Advancement of Artificial Intelligence*, Ocotober 2008.

[11] D. Friedman, Y. Feldman, A. Shamir, and T. Dagan. Automated creation of movie summaries in interactive virtual environments. In *Proceedings of IEEE Virtual Reality*, pages 191–199, 2004.

[12] N. Halper and M. Masuch. Action summary for computer games: Extracting action for spectator modes and summaries. In *Proc. of 2nd Intl Conf. Application and Development of Computer Games*, pages 124–132. Citeseer, 2003.

[13] Nicolas Halper and Patrick Olivier. Camplan: A camera planning agent. In *In Proceedings 2000 AAAI Spring Symposium Series on Smart Graphics*, pages 92–100, March 2000.

[14] Chia-Jung Chan, Ruck Thawonmas, and Kuan-Ta Chen. Automatic storytelling in comics: a case study on world of warcraft. In *CHI EA '09: Proceedings of the 27th international conference extended abstracts on Human factors in computing systems*, pages 3589–3594, New York, NY, USA, 2009. ACM.

[15] A. Shamir, M. Rubinstein, and T. Levinboim. Generating comics from 3d interactive computer graphics. *IEEE Computer Graphics and Applications*, pages 53–61, 2006.

[16] Jinjun Wang, Changsheng Xu, Engsiong Chng, Kongwah Wah, and Qi Tian. Automatic replay generation for soccer video broadcasting. In *MULTIMEDIA '04: Proceedings of the 12th annual ACM international conference on Multimedia*, pages 32–39, New York, NY, USA, 2004. ACM.

[17] Huaxin Xu and Tat-Seng Chua. The fusion of audio-visual features and external knowledge for event detection in team sports video. In *MIR '04: Proceedings of the 6th ACM SIGMM international workshop on Multimedia information retrieval*, pages 127–134, New York, NY, USA, 2004. ACM.

[18] *Half-Life 2*. Valve Corporation, 2004.

[19] L. Nacke, C. Lindley, and S. Stellmach. Log whos playing: psychophysiological game analysis made easy through event logging. In *International conference on Fun and Games*. Springer, 2008.

[20] Luca Galli, Daniele Loiacono, and Pier Luca Lanzi. Learning a context-aware weapon selection policy for unreal tournament iii. In *IEEE Symposium on Computational Intelligence and Games*, 2009.

[21] Matthew P. Sheehan. General agent learning using first person shooter game logs. Master's thesis, The University of Auckland, 2008.

[22] Thomas James Lodato. Spacecam: A tool for machinima cinematography and editing. Unpublished Manuscript, Georgia Institute of Technology, http://lcc.gatech.edu/ tlodato3/Machinima/reading/Lodato_SpaceCam_postmortem.pdf.

[23] *Unreal Tournament 2004*. Epic Games, 2004.

[24] David K. Elson and Mark O. Reidl. A lightweight intelligent virtual cinematography system for machinima production. In *In Proceedings of the 3rd Conference on AI for Interactive Digital Entertainment*, 2007.

[25] Steven M. Drucker and David Zeltzer. Camdroid: A system for implementing intelligent camera control. In *SI3D '95: Proceedings of the 1995 symposium on Interactive 3D graphics*, pages 139–144, New York, NY, USA, 1995. ACM.

[26] William H. Bares, Joël P. Grégoire, and James C. Lester. Realtime constraint-based cinematography for complex interactive 3d worlds. In *AAAI '98/IAAI '98: Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence*, pages 1101–1106, Menlo Park, CA, USA, 1998. American Association for Artificial Intelligence.

[27] David B. Christianson, Sean E. Anderson, Li wei He, David Salesin, Daniel S. Weld, and Michael F. Cohen. Declarative camera control for automatic cinematography. In *In Proceedings of the Thirteenth National Conference on Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conference*, pages 148–155, August 1996.

[28] Li-wei He, Michael F. Cohen, and David H. Salesin. The virtual cinematographer: A paradigm for automatic real-time camera control and directing. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 217–224, New York, NY, USA, 1996. ACM.

[29] Daniel Amerson and Shaun Kime. Real-time cinematic camera control for interactive narratives. In *Working notes of AAAI Spring Symposium*, Stanford, CA, 2000.

[30] Arnav Jhala. An intelligent camera planning system for dynamic narratives. Master's thesis, North Carolina State University, 2004.

[31] Arnav Jhala. *Cinematic Discourse Generation.* PhD thesis, North Carolina State University, 2009.

[32] Jeremy Vineyard. *Setting Up Your Shots: Great Camera Moves Every Filmmaker Should Know.* Michael Wiese Productions, 2000.

[33] Beepa Pty Ltd. Fraps. http://www.fraps.com/.

[34] Avisynth. http://avisynth.org/.

[35] Bruce Mamer. *Film Production Technique: Creating the Accomplished Image.* Thompson Wadsworth, 4th edition, 2006.

[36] Avery Lee. Virtualdub. http://www.virtualdub.org/.

[37] Peter Dragon. Programmatically uploading videos to youtube. http://trailsinthesand.com/programmatically-uploading-videos-to-youtube/.

[38] *24*. FOX, 2001.

Appendices

APPENDIX A

```
<ATMessages>
  <ATMessage>
    <type>startgame</type>
    <timestamp>0.0000</timestamp>
    <ID>0</ID>
  </ATMessage>
  <ATMessage>
    <type>flagpickup</type>
    <playerName>ATPlayerController_0</playerName>
    <playerPawn>ATPawn_2</playerPawn>
    <playerTeamColor>Red</playerTeamColor>
    <flagName>AT_CTFFlag_Blue_0</flagName>
    <timestamp>23.5101</timestamp>
    <ID>17</ID>
  </ATMessage>
  <ATMessage>
    <type>flagpickup</type>
    <playerName>ATBot_0</playerName>
    <playerPawn>ATPawn_1</playerPawn>
    <playerTeamColor>Blue</playerTeamColor>
    <flagName>AT_CTFFlag_Red_0</flagName>
    <timestamp>33.7919</timestamp>
    <ID>21</ID>
  </ATMessage>
  <ATMessage>
    <type>beginfire</type>
    <weapon>AT Stinger</weapon>
    <playerName>ATBot_0</playerName>
    <playerPawn>ATPawn_1</playerPawn>
    <playerTeamColor>Blue</playerTeamColor>
    <timestamp>41.4549</timestamp>
    <ID>32</ID>
  </ATMessage>
```


Figure A.2: An example set of log messages sent from a Capture the Flag match in *Unreal Tournament 3*. The timestamps refer to the number of seconds that have passed since the start of the match. Some messages from this game session were removed for conciseness. Continued on next page.

```
<ATMessage>
  <type>damage</type>
  <instigatorName>ATPlayerController_0</instigatorName>
  <instigatorPawn>ATPawn_2</instigatorPawn>
  <instigatorTeamColor>Red</instigatorTeamColor>
  <victimName>ATBot_0</victimName>
  <victimPawn>ATPawn_1</victimPawn>
  <victimTeamColor>Blue</victimTeamColor>
  <damagetype>UTDmgType_Rocket</damagetype>
  <amount>80</amount>
  <timestamp>41.7211</timestamp>
  <ID>33</ID>
</ATMessage>
<ATMessage>
  <type>damage</type>
  <instigatorName>ATBot_0</instigatorName>
  <instigatorPawn>ATPawn_1</instigatorPawn>
  <instigatorTeamColor>Blue</instigatorTeamColor>
  <victimName>ATPlayerController_0</victimName>
  <victimPawn>ATPawn_2</victimPawn>
  <victimTeamColor>Red</victimTeamColor>
  <damagetype>UTDmgType_StingerBullet</damagetype>
  <amount>5</amount>
  <timestamp>43.4604</timestamp>
  <ID>35</ID>
</ATMessage>
<ATMessage>
  <type>kill</type>
  <killerName>ATPlayerController_0</killerName>
  <killerPawn>ATPawn_2</killerPawn>
  <killerTeamColor>Red</killerTeamColor>
  <victimName>ATBot_0</victimName>
  <victimPawn>ATPawn_1</victimPawn>
  <victimTeamColor>Blue</victimTeamColor>
  <damagetype>UTDmgType_Rocket</damagetype>
  <timestamp>44.4369</timestamp>
  <ID>36</ID>
</ATMessage>
```


Figure A.2: (Continued) An example set of log messages sent from a Capture the Flag match in *Unreal Tournament 3*. Continued on next page.

```
  <ATMessage>
    <type>flagdrop</type>
    <playerName>ATBot_0</playerName>
    <playerPawn>ATPawn_1</playerPawn>
    <playerTeamColor>Blue</playerTeamColor>
    <flagName>AT_CTFFlag_Red_0</flagName>
    <timestamp>44.4369</timestamp>
    <ID>38</ID>
  </ATMessage>
  <ATMessage>
    <type>flagreturning</type>
    <playerName>ATPlayerController_0</playerName>
    <playerPawn>ATPawn_2</playerPawn>
    <playerTeamColor>Red</playerTeamColor>
    <flagName>AT_CTFFlag_Red_0</flagName>
    <timestamp>45.9140</timestamp>
    <ID>45</ID>
  </ATMessage>
  <ATMessage>
    <type>flagscore</type>
    <playerName>ATPlayerController_0</playerName>
    <playerPawn>ATPawn_2</playerPawn>
    <playerTeamColor>Red</playerTeamColor>
    <flagName>AT_CTFFlag_Red_0</flagName>
    <timestamp>53.6511</timestamp>
    <ID>49</ID>
  </ATMessage>
  <ATMessage>
    <type>endgame</type>
    <timestamp>67.7258</timestamp>
    <ID>54</ID>
  </ATMessage>
</ATMessages>
```


Figure A.2: (Continued) An example set of log messages sent from a Capture the Flag match in *Unreal Tournament 3*.

```
<camera-actions>
  <camera-action>
    <file-name>CamAction-FollowPlayerFrontLong</file-name>
    <variables>
      <timestamps>
        <timestamp>
          <name>start-time</name>
          <value>[timestamp]-9</value>
        </timestamp>
        <timestamp>
          <name>end-time</name>
          <value>[timestamp]-6</value>
        </timestamp>
      </timestamps>
      <actors>
        <actor>
          <name>target</name>
          <value>[playerPawn]</value>
        </actor>
      </actors>
    </variables>
    <constraints>
      <constraint>
        <name>importance</name>
        <value>1</value>
      </constraint>
    </constraints>
  </camera-action>
```

Figure A.3: The camera action series used for the flag pickup action with the player as the focus. Continued on next page.

```
<camera-action>
  <file-name>CamAction-FollowPlayerCloseup</file-name>
  <variables>
    <timestamps>
      <timestamp>
        <name>start-time</name>
        <value>[timestamp]-6</value>
      </timestamp>
      <timestamp>
        <name>end-time</name>
        <value>[timestamp]-3</value>
      </timestamp>
      <timestamp>
        <name>transition-end-time</name>
        <value>[timestamp]-5</value>
      </timestamp>
    </timestamps>
    <actors>
      <actor>
        <name>target</name>
        <value>[playerPawn]</value>
      </actor>
    </actors>
  </variables>
  <constraints>
    <constraint>
      <name>importance</name>
      <value>1</value>
    </constraint>
  </constraints>
</camera-action>
```

Figure A.3: (Continued) The camera action series used for the flag pickup action with the player as the focus. Continued on next page.

```
  <camera-action>
    <file-name>CamAction-InLineWithTwoTargets</file-name>
    <variables>
      <timestamps>
        <timestamp>
          <name>start-time</name>
          <value>[timestamp]-3</value>
        </timestamp>
        <timestamp>
          <name>end-time</name>
          <value>[timestamp]+3</value>
        </timestamp>
        <timestamp>
          <name>transition-end-time</name>
          <value>[timestamp]-2</value>
        </timestamp>
      </timestamps>
      <actors>
        <actor>
          <name>target</name>
          <value>[playerPawn]</value>
        </actor>
        <actor>
          <name>target2</name>
          <value>[flagName]</value>
        </actor>
      </actors>
    </variables>
    <constraints>
      <constraint>
        <name>must-include-timestamp</name>
        <value>[timestamp]</value>
      </constraint>
      <constraint>
        <name>importance</name>
        <value>10</value>
      </constraint>
    </constraints>
  </camera-action>
</camera-actions>
```

Figure A.3: (Continued) The camera action series used for the flag pickup action with the player as the focus.

```
<camera-actions>
  <camera-action>
    <file-name>CamAction-FollowPlayerFrontLong</file-name>
    <variables>
      <timestamps>
        <timestamp>
          <name>start-time</name>
          <value>[timestamp]-6</value>
        </timestamp>
        <timestamp>
          <name>end-time</name>
          <value>[timestamp]-3</value>
        </timestamp>
      </timestamps>
      <actors>
        <actor>
          <name>target</name>
          <value>[killerPawn]</value>
        </actor>
      </actors>
    </variables>
    <constraints>
      <constraint>
        <name>importance</name>
        <value>1</value>
      </constraint>
    </constraints>
  </camera-action>
```

Figure A.4: The camera action series used for the kill action with the killer as the focus. Continued on next page.

```
<camera-action>
  <file-name>CamAction-InLineWithTwoTargets</file-name>
  <variables>
    <timestamps>
      <timestamp>
        <name>start-time</name>
        <value>[timestamp]-3</value>
      </timestamp>
      <timestamp>
        <name>end-time</name>
        <value>[timestamp]+3</value>
      </timestamp>
      <timestamp>
        <name>transition-end-time</name>
        <value>[timestamp]-2</value>
      </timestamp>
    </timestamps>
    <actors>
      <actor>
        <name>target</name>
        <value>[killerPawn]</value>
      </actor>
      <actor>
        <name>target2</name>
        <value>[victimPawn]</value>
      </actor>
    </actors>
  </variables>
  <constraints>
    <constraint>
      <name>must-include-timestamp</name>
      <value>[timestamp]</value>
    </constraint>
    <constraint>
      <name>importance</name>
      <value>10</value>
    </constraint>
  </constraints>
</camera-action>
```

Figure A.4: (Continued) The camera action series used for the kill action with the killer as the focus. Continued on next page.

```
  <camera-action>
    <file-name>CamAction-FollowPlayerBackLow</file-name>
    <variables>
      <timestamps>
        <timestamp>
          <name>start-time</name>
          <value>[timestamp]+3</value>
        </timestamp>
        <timestamp>
          <name>end-time</name>
          <value>[timestamp]+6</value>
        </timestamp>
        <timestamp>
          <name>transition-end-time</name>
          <value>[timestamp]+6</value>
        </timestamp>
      </timestamps>
      <actors>
        <actor>
          <name>target</name>
          <value>[killerPawn]</value>
        </actor>
      </actors>
    </variables>
    <constraints>
      <constraint>
        <name>importance</name>
        <value>1</value>
      </constraint>
    </constraints>
  </camera-action>
</camera-actions>
```

Figure A.4: (Continued) The camera action series used for the kill action with the killer as the focus.

```
<camera-actions>
  <camera-action>
    <file-name>CamAction-FollowPlayerFrontLong</file-name>
    <variables>
      <timestamps>
        <timestamp>
          <name>start-time</name>
          <value>[timestamp]-9</value>
        </timestamp>
        <timestamp>
          <name>end-time</name>
          <value>[timestamp]-6</value>
        </timestamp>
      </timestamps>
      <actors>
        <actor>
          <name>target</name>
          <value>[playerPawn]</value>
        </actor>
      </actors>
    </variables>
    <constraints>
      <constraint>
        <name>importance</name>
        <value>1</value>
      </constraint>
    </constraints>
  </camera-action>
```

Figure A.5: The camera action series used for the flag score action with the player as the focus. Continued on next page.

```
<camera-action>
  <file-name>CamAction-FollowPlayerCloseup</file-name>
  <variables>
    <timestamps>
      <timestamp>
        <name>start-time</name>
        <value>[timestamp]-6</value>
      </timestamp>
      <timestamp>
        <name>end-time</name>
        <value>[timestamp]-3</value>
      </timestamp>
      <timestamp>
        <name>transition-end-time</name>
        <value>[timestamp]-5</value>
      </timestamp>
    </timestamps>
    <actors>
      <actor>
        <name>target</name>
        <value>[playerPawn]</value>
      </actor>
    </actors>
  </variables>
  <constraints>
    <constraint>
      <name>importance</name>
      <value>1</value>
    </constraint>
  </constraints>
</camera-action>
```

Figure A.5: (Continued) The camera action series used for the flag score action with the player as the focus. Continued on next page.

```
  <camera-action>
    <file-name>CamAction-InLineWithTwoTargets</file-name>
    <variables>
      <timestamps>
        <timestamp>
          <name>start-time</name>
          <value>[timestamp]-3</value>
        </timestamp>
        <timestamp>
          <name>end-time</name>
          <value>[timestamp]+3</value>
        </timestamp>
        <timestamp>
          <name>transition-end-time</name>
          <value>[timestamp]-2</value>
        </timestamp>
      </timestamps>
      <actors>
        <actor>
          <name>target</name>
          <value>[playerPawn]</value>
        </actor>
        <actor>
          <name>target2</name>
          <value>[flagName]</value>
        </actor>
      </actors>
    </variables>
    <constraints>
      <constraint>
        <name>must-include-timestamp</name>
        <value>[timestamp]</value>
      </constraint>
      <constraint>
        <name>importance</name>
        <value>10</value>
      </constraint>
    </constraints>
  </camera-action>
</camera-actions>
```

Figure A.5: (Continued) The camera action series used for the flag score action with the player as the focus.

```
<camera-events>
  <camera-event narrativeName="capturetheflag" narrativeRank="-1"
                narrativeID="1">
    <action>
      <type>ATAction_FastForward</type>
      <start-time>0</start-time>
      <end-time>0</end-time>
      <spawn-pawn-name>ATPawn_2</spawn-pawn-name>
      <spawn-time>1.6548</spawn-time>
    </action>
    <action>
      <type>ATAction_StartCapture</type>
      <start-time>14.5101</start-time>
      <end-time>14.5101</end-time>
    </action>
    <camera-action>
      <timestamps>
        <start-time>14.5101</start-time>
        <end-time>17.5101</end-time>
        <transition-end-time>-1</transition-end-time>
      </timestamps>
      <camera-modifiers>
        <camera-modifier>
          <start-time>14.5101</start-time>
          <end-time>17.5101</end-time>
          <target>ATPawn_2</target>
          <type>ATCamMod_LocationRelative</type>
          <direction-yaw>0</direction-yaw>
          <direction-pitch>0</direction-pitch>
          <direction-roll>0</direction-roll>
          <distance>100</distance>
        </camera-modifier>
```


Figure A.6: The camera event generated by using the camera action series show in Figure A.3, A.4, and A.5 with the narrative with ID = 1 in Figure 3.6 and the log messages from A.2. Continued on next page.

```
      <camera-modifier>
        <start-time>14.5101</start-time>
        <end-time>17.5101</end-time>
        <target>ATPawn_2</target>
        <type>ATCamMod_RotationRelative</type>
        <yaw>180</yaw>
        <pitch>0</pitch>
        <roll>0</roll>
      </camera-modifier>
      <camera-modifier>
        <start-time>14.5101</start-time>
        <end-time>17.5101</end-time>
        <type>ATCamMod_FOV</type>
        <FOV>90</FOV>
      </camera-modifier>
    </camera-modifiers>
  </camera-action>
  <camera-action>
    <timestamps>
      <start-time>17.5101</start-time>
      <end-time>20.5101</end-time>
      <transition-end-time>18.5101</transition-end-time>
    </timestamps>
    <camera-modifiers>
      <camera-modifier>
        <start-time>17.5101</start-time>
        <end-time>20.5101</end-time>
        <target>ATPawn_2</target>
        <type>ATCamMod_LocationRelative</type>
        <direction-yaw>0</direction-yaw>
        <direction-pitch>30</direction-pitch>
        <direction-roll>0</direction-roll>
        <distance>55</distance>
      </camera-modifier>
```


Figure A.6: (Continued) Example camera event. Continued on next page.

```
      <camera-modifier>
        <start-time>17.5101</start-time>
        <end-time>20.5101</end-time>
        <target>ATPawn_2</target>
        <bone>b_head</bone>
        <type>ATCamMod_RotationLookAt</type>
      </camera-modifier>
      <camera-modifier>
        <start-time>17.5101</start-time>
        <end-time>20.5101</end-time>
        <type>ATCamMod_FOV</type>
        <FOV>45</FOV>
      </camera-modifier>
    </camera-modifiers>
  </camera-action>
  <camera-action>
    <timestamps>
      <start-time>20.5101</start-time>
      <end-time>26.5101</end-time>
      <transition-end-time>21.5101</transition-end-time>
    </timestamps>
    <camera-modifiers>
      <camera-modifier>
        <type>ATCamMod_BulletTime</type>
        <start-time>20.5101</start-time>
        <starting-speed>.75</starting-speed>
        <stop-slowing-down-time>
        23.433176923078</stop-slowing-down-time>
        <start-speeding-up-time>
        23.587023076922</start-speeding-up-time>
        <ending-speed>.75</ending-speed>
        <end-time>26.5101</end-time>
        <target-speed>0.204081632653003</target-speed>
      </camera-modifier>
```


Figure A.6: (Continued) Example camera event. Continued on next page.

```
      <camera-modifier>
        <type>ATCamMod_InLineWithTwoTargets</type>
        <start-time>20.5101</start-time>
        <end-time>26.5101</end-time>
        <target>ATPawn_2</target>
        <target2>AT_CTFFlag_Blue_0</target2>
        <distance>100</distance>
        <z-offset>75</z-offset>
        <adjust-yaw>0</adjust-yaw>
        <adjust-pitch>-20</adjust-pitch>
        <adjust-roll>0</adjust-roll>
      </camera-modifier>
      <camera-modifier>
        <start-time>20.5101</start-time>
        <end-time>26.5101</end-time>
        <type>ATCamMod_FOV</type>
        <FOV>90</FOV>
      </camera-modifier>
    </camera-modifiers>
  </camera-action>
  <action>
    <type>ATAction_StopCapture</type>
    <start-time>26.5101</start-time>
    <end-time>26.5101</end-time>
  </action>
  <action>
    <type>ATAction_StartCapture</type>
    <start-time>38.4369</start-time>
    <end-time>38.4369</end-time>
  </action>
```


Figure A.6: (Continued) Example camera event. Continued on next page.

```
<camera-action>
  <timestamps>
    <start-time>38.4369</start-time>
    <end-time>41.4369</end-time>
    <transition-end-time>-1</transition-end-time>
  </timestamps>
  <camera-modifiers>
    <camera-modifier>
      <start-time>38.4369</start-time>
      <end-time>41.4369</end-time>
      <target>ATPawn_2</target>
      <type>ATCamMod_LocationRelative</type>
      <direction-yaw>0</direction-yaw>
      <direction-pitch>0</direction-pitch>
      <direction-roll>0</direction-roll>
      <distance>100</distance>
    </camera-modifier>
    <camera-modifier>
      <start-time>38.4369</start-time>
      <end-time>41.4369</end-time>
      <target>ATPawn_2</target>
      <type>ATCamMod_RotationRelative</type>
      <yaw>180</yaw>
      <pitch>0</pitch>
      <roll>0</roll>
    </camera-modifier>
    <camera-modifier>
      <start-time>38.4369</start-time>
      <end-time>41.4369</end-time>
      <type>ATCamMod_FOV</type>
      <FOV>90</FOV>
    </camera-modifier>
  </camera-modifiers>
</camera-action>
```


Figure A.6: (Continued) Example camera event. Continued on next page.

```
<camera-action>
  <timestamps>
    <start-time>41.4369</start-time>
    <end-time>47.4369</end-time>
    <transition-end-time>42.4369</transition-end-time>
  </timestamps>
  <camera-modifiers>
    <camera-modifier>
      <type>ATCamMod_BulletTime</type>
      <start-time>41.4369</start-time>
      <starting-speed>.75</starting-speed>
      <stop-slowing-down-time>
      44.359976923078</stop-slowing-down-time>
      <start-speeding-up-time>
      44.513823076922</start-speeding-up-time>
      <ending-speed>.75</ending-speed>
      <end-time>47.4369</end-time>
      <target-speed>0.204081632653003</target-speed>
    </camera-modifier>
    <camera-modifier>
      <type>ATCamMod_InLineWithTwoTargets</type>
      <start-time>41.4369</start-time>
      <end-time>47.4369</end-time>
      <target>ATPawn_2</target>
      <target2>ATPawn_1</target2>
      <distance>100</distance>
      <z-offset>75</z-offset>
      <adjust-yaw>0</adjust-yaw>
      <adjust-pitch>-20</adjust-pitch>
      <adjust-roll>0</adjust-roll>
    </camera-modifier>
    <camera-modifier>
      <start-time>41.4369</start-time>
      <end-time>47.4369</end-time>
      <type>ATCamMod_FOV</type>
      <FOV>90</FOV>
    </camera-modifier>
  </camera-modifiers>
</camera-action>
```


Figure A.6: (Continued) Example camera event. Continued on next page.

```
<action>
  <type>ATAction_StopCapture</type>
  <start-time>47.4369</start-time>
  <end-time>47.4369</end-time>
</action>
<action>
  <type>ATAction_StartCapture</type>
  <start-time>47.6511</start-time>
  <end-time>47.6511</end-time>
</action>
<camera-action>
  <timestamps>
    <start-time>47.6511</start-time>
    <end-time>50.6511</end-time>
    <transition-end-time>-1</transition-end-time>
  </timestamps>
  <camera-modifiers>
    <camera-modifier>
      <start-time>47.6511</start-time>
      <end-time>50.6511</end-time>
      <target>ATPawn_2</target>
      <type>ATCamMod_LocationRelative</type>
      <direction-yaw>0</direction-yaw>
      <direction-pitch>30</direction-pitch>
      <direction-roll>0</direction-roll>
      <distance>55</distance>
    </camera-modifier>
    <camera-modifier>
      <start-time>47.6511</start-time>
      <end-time>50.6511</end-time>
      <target>ATPawn_2</target>
      <bone>b_head</bone>
      <type>ATCamMod_RotationLookAt</type>
    </camera-modifier>
    <camera-modifier>
      <start-time>47.6511</start-time>
      <end-time>50.6511</end-time>
      <type>ATCamMod_FOV</type>
      <FOV>45</FOV>
    </camera-modifier>
  </camera-modifiers>
</camera-action>
```


Figure A.6: (Continued) Example camera event. Continued on next page.

```
<camera-action>
  <timestamps>
    <start-time>50.6511</start-time>
    <end-time>56.6511</end-time>
    <transition-end-time>51.6511</transition-end-time>
  </timestamps>
  <camera-modifiers>
    <camera-modifier>
      <type>ATCamMod_BulletTime</type>
      <start-time>50.6511</start-time>
      <starting-speed>.75</starting-speed>
      <stop-slowing-down-time>
      53.574176923078</stop-slowing-down-time>
      <start-speeding-up-time>
      53.728023076922</start-speeding-up-time>
      <ending-speed>.75</ending-speed>
      <end-time>56.6511</end-time>
      <target-speed>0.204081632653003</target-speed>
    </camera-modifier>
    <camera-modifier>
      <type>ATCamMod_InLineWithTwoTargets</type>
      <start-time>50.6511</start-time>
      <end-time>56.6511</end-time>
      <target>ATPawn_2</target>
      <target2>AT_CTFFlag_Red_0</target2>
      <distance>100</distance>
      <z-offset>75</z-offset>
      <adjust-yaw>0</adjust-yaw>
      <adjust-pitch>-20</adjust-pitch>
      <adjust-roll>0</adjust-roll>
    </camera-modifier>
    <camera-modifier>
      <start-time>50.6511</start-time>
      <end-time>56.6511</end-time>
      <type>ATCamMod_FOV</type>
      <FOV>90</FOV>
    </camera-modifier>
  </camera-modifiers>
</camera-action>
```


Figure A.6: (Continued) Example camera event. Continued on next page.

```
    <action>
      <type>ATAction_StopCapture</type>
      <start-time>56.6511</start-time>
      <end-time>56.6511</end-time>
    </action>
    <action>
      <type>ATAction_CompileVideo</type>
      <start-time>56.6511</start-time>
      <end-time>56.6511</end-time>
    </action>
  </camera-event>
</camera-events>
```

Figure A.6: (Continued) Example camera event.

APPENDIX B

Unreal Tournament 3 - Capture the Flag

You will be playing in a team-based Capture the Flag match in the first-person shooter Unreal Tournament 3.

Capture the Flag Rules

- Your objective is to take the opposing team's flag and bring it back to your own flag's base. The Red team must capture the Blue flag while protecting the Red flag. The Blue team must capture the Red flag while protecting the Blue flag.
- To pick up the opposing team's flag, simply move over it.
- You drop the flag if you are killed.
- If an opponent is killed and drops your flag, move over it to return it to your base.
- If a teammate is killed and drops the flag, move over it to pick it up and continue to your base to score.
- The first team to score 3 times wins.
- Note that if the opposing team has taken your flag, you cannot score until it is returned!

Unreal Tournament 3 Controls

Action	Key
Move Forward	W
Move Backward	S
Strafe Left	A
Strafe Right	D
Jump	Spacebar
Crouch	C
Fire	Mouse Left Click
Alternate Fire (Zoom, burst fire, etc)	Mouse Right Click
Previous Weapon	Mouse Scroll Up
Next Weapon	Mouse Scroll Down
Navigation Help (Shows path to objective)	F2

Figure B.1: The first page of the reference sheet given to the participants in the preliminary evaluation, describing the game's rules and controls.

Capture the Flag HUD

Top – The number of Red and Blue team captured flags is displayed. White arrows point the way to each team's flag base. Displays an icon indicating both teams' flag status. There are icons for Secure, Taken, and Dropped.

Bottom Left – Shows your health and armor.

Bottom Middle – Shows the weapons you have available.

Bottom Right – Shows amount of ammunition available for the weapon you have equipped.

Bottom – The HUD background color is your team color (Red in this example).

Figure B.2: The second page of the reference sheet given to the participants in the preliminary evaluation, describing the game's heads up display.

Gameplay Survey

Character Name (**NOT YOUR REAL NAME**): _____

1. Please describe three moments you found to be most exciting, humorous, interesting, or notable during gameplay. Be sure to note the time or range of time that the moment occurred.

    a. Time: _____ to _____

    Description:

    b. Time: _____ to _____

    Description:

    c. Time: _____ to _____

    Description:

2. Did your team win, lose, or tie in the Capture the Flag match?
    a. Won
    b. Lost
    c. Tied
3. Before today's gameplay session, how experienced were you with first person shooters?
    a. Not experienced at all.
    b. Somewhat experienced.
    c. Very experienced.
4. Before today's gameplay session, how experienced were you playing Capture the Flag matches in Unreal Tournament 3?
    a. Not experienced at all.
    b. Somewhat experienced.
    c. Very experienced.

Figure B.3: The survey completed by each participant in the preliminary evaluation as they watched the replay of the match they just played.

Video Survey

Character Name (**NOT YOUR REAL NAME**): _____ Video #: _____

1. Please describe the events that took place in the video.

2. Please rate this video on the following criteria. Circle one number on each row.

Not Exciting	1	2	3	4	5	6	7	Very Exciting
Not Humorous	1	2	3	4	5	6	7	Very Humorous
Not Dramatic	1	2	3	4	5	6	7	Very Dramatic
Not Interesting	1	2	3	4	5	6	7	Very Interesting
Not Coherent	1	2	3	4	5	6	7	Very Coherent

3. When you were playing the game did you witness or participate in any of the depicted events in the video? **CIRCLE ONE**
   a. Did not witness or participate in any of the events.
   b. I witnessed or participated in some of the events.
   c. I witnessed or participated in all of the events.
   d. I can't remember if I witnessed or participated in the events.

4. If you answered (b) or (c) to question 3, did you describe part or all of the events in this video in your gameplay survey? If part, please describe which parts.

5. What did you like about the video? Why?

6. What did you dislike about the video? Why?

7. Please write any other comments you have in regards to the video.

Figure B.4: The survey completed by each participant in the preliminary evaluation for each video generated from their gameplay session.