INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa

AutoBroad: A Realtime Broadcasting System for E-Sports

Hugo Damas

Dissertação para obtenção do Grau de Mestre em
Engenharia Informática e de Computadores

Júri

Presidente:	Prof João Marques Silva
Orientador:	Prof Carlos Martinho
Vogais:	Prof João Pereira

November 2011

Acknowledgements

Acknowledgements to my Prof. Carlos Martinho, without whom the very concept of this thesis would not be a reality, and to IST itself, which has trained me into a knowledgeable and efficient computer engineer. Finally, acknowledgements to my colleagues, with whom I developed the indie video game on which this thesis was implemented.
All three were instrumental in making the development, and especially the testing, of this thesis possible.

Oeiras, November 23, 2011
Hugo Damas

I dedicate this work to my parents, who not only raised me, but paid all the expenses of my education without ever asking anything back but a worthwhile commitment to my studies. As I reach a mighty stepping stone in my academic career, it is my sincerest hope that I have made them proud, and that my success is worth all the effort and dedication that they have put into me.

Resumo

Neste documento, iremos entrar na industria de desportos electrónicos.
Mostraremos como investigámos e construimos o que é o primeiro passo para construir um sistema automático de emissão de partidas de jogos multi-jogador, que seja automático e completamente funcional, capaz de assistir comentadores com o emissões de jogos multi-jogador ao vivo.
Nós olhámos para a industria de emissão de desportos eletronicos, observando as suas tácticas e dificuldades, conseguindo então desenvolver uma solução que acreditamos ser capaz de mostrar a um espectador suficientes eventos relevantes, durante uma partida, correctamente, que o espectador será capaz de completamente entender a cadeia de causa e efeito por trás dos diferentes estados competitivos da partida.
Acreditamos que este é um primeiro passo, valioso, em direcção a um sistema de emissão em tempo real aperfeiçoado, que possa ser desenvolvido no futuro.

Abstract

In this document, we delve into the world of sports and E-Sports broadcasting.
We will show how we investigated and built what is the first step into building a fully functional real-time automatic broadcasting system, that may assist commentators with the live broadcasting of e-sports.
We have looked into the industry, observing their tactics and difficulties, and thus have developed a solution we believe is able to show the spectator enough relevant events during the course of game progress, correctly, that the spectator fully understands the chains of cause and effect behind the different competitive states of the match.
We believe this is a worthwhile first step to a much more perfected realtime broadcasting system that may come in the future.

Palavras Chave
Keywords

Palavras Chave

Desportos Electrónicos
Emissão De Desporto
Acessão de Jogo
Acessão de Partida
Partida baseada no Jogador
Sistema em Tempo Real
Sistema Automático
Heurísticas de Emissão de Desporto

Keywords

Electronic Sports
Sports Broadcasting
Game Assessment
Match Assessment
Player-based Match
Realtime System
Automatic System
Sports Broadcasting Heuristics

Índice

List of Figures

List of Tables

Chapter 1

Introduction

What exactly is the problem that we have tackled? Why was it important to be tackled?

And how well did we tackle it? What was our contribution to the matter at hand?

In this introductory section, it is our focus to ease the reader into the domain where this dissertation was developed, as well as give an overview of our contribution to this new field of investigation.

1.1 Eletronic Sports Broadcasting

Electronic Sports is a term directly relating to multi-player video-games that are approached in the exact same way conventional sports are: tournaments, championships and seasonal leagues are made out of them, events which involve matches between individuals or teams. These matches are then broadcasted to viewers around a region, or even internationally, creating fan-bases which in turn generates attention and money towards the game, the individuals or teams that professionally compete, and all entities involved in the hosting of said events.

Broadcasting is the distribution of audio and video content to a dispersed audience via radio, television, or other, often digital transmission media. Receiving parties may include the general public or a relatively large subset thereof.

The broadcasting crew for something like this is most often, if not always, made up of two individuals: the expert and the sports commentator. The sports commentator is someone who is in the business of broadcasting, employed at that function by whatever entity is hosting the match, and is normally studied in the game in question. The expert is normally a player, professional or sufficiently approximate, that is there to give his opinion and overall help the commentator, which will often question the expert about the state of the game.

But these broadcasting teams highly lack decent camera coverage over a video-game match. They both use the spectator mode to overview the game. This spectator mode gives the user a chance to switch through player-views, as well as give an overview of the entire game. But there are no perspectives other than the one native to the game. Most often, if not always, this spectator mode is not developed for broadcasting coverage

but for something other players can use online to pass the time while they are dead or otherwise temporarily unable to play. The idea is the broadcasting crew streams this spectator mode, in real time, to anyone watching. So every spectator is actually watching the stream of a spectator mode being controlled by one of the two broadcasters.

We have envisioned a system that can be developed for any game, a system that will provide an acceptable and fulfilling realtime broadcasting. One that focuses on showing the viewer what is relevant, and does so in an acceptably enough visually appealing way, and in realtime.

But there are a number of different genres of E-sports, which is to say, there are a number of different e-sports. Normally, a video-game represents, by itself, a sport; the same way one plays soccer or American football, one plays Quake or Starcraft.

Like conventional sports, each game (e-sport) has its related ethnicity of popularity and fame, but all in all, it is a notably rising facet of entertainment even for non video-game enthusiasts.

There are several kinds of competitive E-sports, and though we will focus on the genre we have named "Third Person Strategy", TPS, we will also discuss every other type of video-game that we see in the competitive world, mainly to familiarize the reader with terms and concepts we will be referring to during the rest of this document.

1.1.1 Real Time Strategy

RTS (Real Time Strategy) games feature a complexity that can out-reach that of chess or any other regular board game [MM09], due to the requirement of rapidly complex multi-tasking. In them, each player controls a faction, managing resources and the economy of that faction, and building and training military assets as deemed necessary with the final objective of vanquishing all enemy factions.

Starcraft 2[16] is a very popular game, at the moment, of this genre. In it, a player picks one of three factions: Terran, Protoss and Zerg, each with their unique needs of management, and ways to defeat the opponent. For example, Protoss can teleport units, Terran can fly units and drop them on locations, and Zerg can tunnel from an entry point to an exit point, even across space.
RTS games are very popular in the eastern scene, where there are even channels dedicated to ongoing seasons of matches between professional players, an actual league. It is very popular in South Korea and with a notably growing popularity worldwide.

1.1.2 Third Person Strategy

TPS (Third Person Strategy), is a term we are coining for this new string of games that started coming out with popular Warcraft 3 mod Defence of the Ancients (DoTA)[17]. It spawned later games of the same genre, League of Legends[14] and Heroes of Newerth[13], produced by the same minds of DoTA. And more are spawning by the year.

In these games, players play a character they can pick from a very large array of characters, which has a very particular and individual set of skills and stats. There is a very wide array of items, with a very wide and varyingly unique array of effects, that

one can buy during the game, and then use the character to play on a map. And though hostility between characters is allowed, it is not the objective of the game. The objective is to conquer strategic points of the opposing team, normally by destruction.

On the games in question, the objective is to destroy a particular building of the opposing team, and to do that, one must destroy defending towers along the way, which will be defended by the opposing team, which have mirrored objecties.

There are computer controlled characters which fight for both sides, much weaker characters, and neutral creatures that are there to give resources or apply modifiers to the characters, which level up as the game progresses. Every player has to carefully manage his income, which he gets from killing anything, in order to get the items he wants, at specific timings according to the state of the game. Every player has to work with his allies, as a team, in order to fight as a team, and to control the map as a team. Very strategic, but so player-based it completely sets it apart from RTS army-based gameplay, which prompted us to use the term TPS to refer to this unique genre. One on which our test-game was largely based out off.

Currently, American and European players are dominating official competitions, but these games are new and largely growing in interest in the eastern scene, namely China, South Korea and Taiwan.

A study on the rest of the video-game genres that represent professional gaming can be found on the Appendix, sections 6.3 through 6.6. These include the FPS(First Person Shooter), TPA(Third Person Action), Beat 'Em Ups and Sports videogames. Some concepts and terms discussed further ahead can be found on those sections of the Appendix.

1.2 A Closer Look at E-Sports Broadcasting Today

In E-Sports, we have observed that the tactic to deal with broadcasting almost always sums up to having two experts dealing with everything. One focused on commentating, the other focused on what is happening in the game. Both do both, but each focuses on his responsability.

It works...for players. Right now, E-sports broadcasting is largely focused and aimed at players of the game in question. This is something actual sports do not do. They do not expect, nor hope, that only individuals who practice the sport will be fans of the sport. Even chess or poker championships go to some lengths to be appealing to non-players, and in conventional sports, this is done by the broadcasting crew, as we have explained above.

Indeed, the fact that commentators are always explaining what is happening, what the players may be thinking or deciding, and spend time and effort explaining basic things only non-players are unfamiliar with, shows that this concern is real and already taken into consideration by some broadcasting crews. But we are specifically referring to camera work.

In FPS and TPA games, for example, they skip between player-cams only, trying to show something other than "he lost and he won", which is hard to tell, for non-experts.

Because in these two genres, decisions are being made and carried out faster than the experts can follow; things happen too fast and one loses a lot trying to find what player should be showed. For example, a Quake Live player hardly ever stays still, or focuses his sights on any given point, for more than a second. There are teleports they often go through back-first, there are jumps they make, also back-first, or side-first. The players know they are about to leap into a completely different set of the map, with a very sudden transition; they do not have to look where they are going, because they know, but viewers do not know what the player is planning. One woul notice this is also a problem with many conventional sports, which is solved by using replay. Unfortunately, the fact is that there is no replay in E-Sports broadcasting.

In RTS and TPS games, we are given a top-down isometric view. In these games, the broadcasting crew actually has some worthwhile camera control; they decide what to show, and even have access to what a team/player can see, so it is less confusing for non players, who may actually be able to follow the action.

The problem is that all players can also look at the entire map, though enemy assets are only visible as long as personal or friendly assets are in proximity. But that means experts will be trying to fool experts, all the while just waiting to react to millisecond events with a millisecond action. The third-party experts, handling the camera, may also easily be fooled. But more than that, an isometric top-down camera is great to see, tactically, what is going on, but we feel it is not ideal to transmit tension, emotion, and specially micro-skills of the player (aim, reaction, distancing, unit control, etc). Commentators on the genre are always forced to spell out these skills.

In Beat 'Em Ups, we are treated to two kinds of camera play: 2D and 3D. 2D features a camera that just pans left and right, focused on showing the two players. 3D features an orbital camera that rotates around the players. The biggest issue here is occlusion, an issue that is directly comparable to fighting sports. 2D does not have much of a problem (though, as an aside, we think it could use a lot from replay or slow-down based camera-work, in order to really entertain a viewer.)

In sports video-games, there is a need to approach them as one would their real counter-part, with the same strategies and techniques. A lot of the work is already done, especially for non racing sports, since they invest a great deal into instant-replay. But we feel our solution may still serve these games well since the user still needs to know at what to look, at any give time.

1.3 The Importance of Broadcasting for the success of a Sport

Broadcasting has long been an important facet of the success of sports. In several examples from literature, like in "Handbook on the economics of sport" by Wladimir Andreff and Stefan Szyman'ski[AS06], it is asserted that the rise to fame, and the economical power of sports, came from the job partaken by broadcasting crews, most notably, by commentators, which are greatly responsible for the emotional impact a broacast can make on the viewer.

The effort of broadcasting teams to relay a match, and the success with which they do

so, through placement of cameras where they need to be to later produce the feedback in realtime, is critical to greatly enhance the enjoyment of the viewers. As a result, sports broadcasting is a great source of income to broadcasting companies, both Television and Radio. More importantly, changes in broadcasting market incur changes in the sports market[Bab06].

So we believe that with E-Sports already experiencing booms of interest and popularity in certain pockets of the international society, it would be greatly beneficial to the industry itself to enhance the broadcasting experience.

This would be the final goal behind the development of the system we have envisioned, and we believe we made a good first step, a leap even, towards that final goal.

1.4 Our Contribution

Our contribution is two-fold: An arquitecture and overall approach to building an automatic broadcasting system for a video-game that is independant of its genre and target demographic. And we have implemented and got it working for a TPS multiplayer video-game.

Regarding the implementation, the system is very light, using only low-level information (like positions, scores, ability cooldown values) and simple math to produce its deliberations, and allows multiple users to log onto the system in order to spectate the game, and watch the match.

The system keeps an update on player competitive status, feeding it to the spectator.

We will be presenting said system in this document.

But before we delve into it directly, we will proceed, in the next part of this document, to first cover prior cases and works that were studied and developed in the scientific community and that we felt were related to ours.

Chapter 2

Related Work

We divided the kinds of related works and cases across three different facets, each having an individual and partial interest to our overall goal and objective, though they can be seen as independent from one another. Plus, a fourth one on studies covering conventional sports broadcasting, since we believed it to be relevant when approaching the development of our system.

So in this part of the document, we will discuss several works which have been developed inside each of those three domains of study and development, discuss about a few works that are, in their whole, conceptually relevant to ours, and then wrap up by discussing works done on conventional sports broadcasting, and what we can take from them.

2.1 Intelligent Camera

The concept of automatic camera is a topic of much issue in the AI community; a camera that, given an object or point to film, will know how to take a shot of it. A shot as in what perspective of the object to show, or what point of view towards the object the camera should assume. More than that, the camera should also avoid occlusion and collision, and even take into consideration close environment in order to make a cinematographical choice when picking the shot.

Quoting the paper on Camplan [HO00]:

"...the stages of the following graphical presentation pipeline characterized in (Seligmann, 1993):

(a) Generation of communicative goal: decide what it is that the image should accomplish.

(b) Selection of presentation strategy: determine the visual effect that will be used to satisfy the communicative goal.

(c) Selection of presentation method: given the presentation strategy specify the different ways by which it may be realized.

(d) Image generation: based upon the selection of the presentation methods, the graphical model of the artifact must be modified to achieve the specified visual properties."

Camplan is a system intended to resolve the last two issues, and so does every other work on intelligent camera design that we have found. Ours differentiates from all of these, because we seek to focus on point (a) and (b), the first two issues.

We will focus on them, for they have not been approached in the domain of Video-Game broadcasting, but we do seek to present a system that solves all four. By applying the most game-genre-specific simplistic solution possible on points (c) and (d), to our solution for (a) and (b), and integrating it all into one single general solution for the domain of real-time video-game broadcasting.

This is what we seek to accomplish and, so far as we know, it has never been attempted by any work on an intelligent camera.

2.2 Information Extraction for an Artificial Intelligence

This is where the interest in this thesis will be more focused. In terms of video-games, since no one has approached a real-time broadcasting system, we looked at the work developed around artificial intelligent characters for games. We looked at the work developed in the area of *strategy prediction.*

Studies have been promising and advancing, throughout a facet of fronts: data-mining pro-level matches to build a case-based processing of low level game data(in RTS, for example, that could be units and buildings); player modeling, in which they interview players to extract strategies and techniques which they implement into the AI player, so it can guess at what the opponent is doing, and perform a counter-tactic; using learning algorithms and training the AI player; and a myriad of other approaches that share a few things that differentiate all of them from ours.

Some real examples one may read, if interested, are "An Evaluation of Models for Predicting Opponent Positions in First-Person Shooter Video Games" [HB], "An Integrated Agent for Playing Real-Time Strategy Games" [MM09], "Rapid Adaptation of Video Game AI" [BSvdH08], "Building a player strategy model by analyzing replays of real-time strategy games" [HS08], and "It Knows What You're Going To Do: Adding Anticipation to a Quakebot" [Lai01].

We will not delve into detail, nor list every other work done in the realm of artificial intelligent characters for games, due to the fact that what differentiates our goal from theirs is applicable to any work in this area: none have tried to be general about their models, in terms of game genre, since there is only a small amount of work done for FPS games and the rest is on RTS games.

Besides being very game-specific, only a handful of techniques and applications have been tested with real players, like it was the case in "Building a player strategy model by analyzing replays of real-time strategy games"[HS08] and "An Evaluation of Models for Predicting Opponent Positions in First-Person Shooter Video Games"[HB]. The rest are tested with scripted players a number of times, in order to derive efficiency against two or three play styles. Successful or not, they are studies aimed at objectively different goals.

Much more importantly, the research is focused on the AI recognizing and forming

strategy and tactics. But they do not seem concerned with recognizing what is relevant to show a spectator, that cause-effect relationship between match-progress related events that is our focus.

They do recognize what is going on in the game, which is still relevant, but only *if the problem is approached in the right manner.* We needed an approach that would translate game-state date into some kind of "context" knowledge, that we could use to recognize interesting events. And we needed the approach to work in a stand-alone manner, which is to say, without any prior knowledge of the match, game-field, and so forth (knowledge one has from data-mining, learning, and every other approach covered by research in strategy prediction).

There is one work, though, that we found to be the exception. And that is because it takes the approach of situation assessment [MOR08]. Situation assessment for plan retrieval in real-time strategy games [MOR08] was a work developed by Mishra, Ontañón and Ram, to try to derive high-level information from low-level data, in order to decide on strategies and how to implement them.

The work was developed on Wargus, a Warcraft 2 Mod that allows one to play Warcraft 2 with the Stratagus engine, which is to say, AI developers have a direct hands-on manipulation with the code of the game engine, and thus can easily put their own AI into the game.

The developed AI in question uses the notion of "situation", interpreted through a pipeline of translations that transform low-level data like *all resources available on map* to high-level information like *distance between two opponents*, into situations.

They then use these situations to filter through case-based models that define strategies and tactics. Only then do they actually choose the strategy and tactics to execute.

Now, we differ in as much that it is not our intent to filter through a list of strategies and tactics, but *through a list of relevant objects to show the viewer.* Also, we will be dealing with a different game genre, and attempt to take an approach that can be generalized to other game genres, but still, this is an approach of the likes we were looking for, and we used it.

2.3 Camera Profiling

We mentioned before that the camera perspective is normally native to the game genre, First Person and top-down isometric view for example. This is not ideal. In order to provide a visually satisfying and enticing experience for a viewer, a more cinematic manipulation of the camera is ideal, if not necessary, one normaly taking into account its target demographic, and its communicative goal.

Battle Cam [YA06], the battle camera implemented on Star Wars: Empire At War TM, was a system presented at the Game Developers Conference in 2005. In it, Yangli Yee and Elie Arabian present a camera system to be activated during RTS battles.

This intelligent camera system picks the most interesting object and then constructs a shot and plays it until it dies, restarting the process.

In this work, their approach was "visual attention", which is a field of study in the

domain of psychology. From it, they drew parameters that could be used to judge objects as more or less interesting.

In the same fashion as us, though, they wanted their system to be light, and thus to only use raw low-level game-state information. So they used size, attack power, location, health, as well as player selection and visibility, to judge the interesting value of the objects to choose from.

Though our approaches are the same, and goals close alike, our work mainly differs in the meaning of the word "interesting". It stands to say that in Battle Cam, "interesting" is actually the more "visually interesting" objects on scene.

We feel our work differs in two ways: first, our system is not custom made for a battle scenario on an RTS game, but rather for *any scenario going on in any video-game*. Second, we seek not objects and/or scenes that are "interesting", but that are "relevant", which is to say, *when they are tactically interesting.*

All in all, visually appealing is a secondary objective, first one being the relevance. Which is why we detach these two objectives (interesting and appealing) into two separate, though compounded, modules: Pick the object, show it. And we have deeply concentrated on the former, not the latter.

Battle cam does this also, but it is a one goal system to demonstrate an overall movie-like experience during a battle. It is, though, something that someone implementing our system could use for RTS games (more on this topic on the future work section of this document).

But the main conclusions we retrieved from researching this domain of study were twofold:

First, camera viewpoints affect the experience of the viewer.

Secondly, different people are differently affected by the same kind of shot [SMY09],[MJY09]. The question is what kind of person should we target?

The answer we arrived at was that the system should thus be genre dependant, because the genre of the game in question already does a good job at splitting types of viewers apart. How one directs an action-comedy production is not how one directs an action-horror production, but the basics are pretty much the same, and applicable. We have sought the basics, not the specializations.

As mentioned in the previous section, intelligent camera design normally comes with "camera profiles", thus most work on this area lays out a preparation for this adaptation, into producing genre-dependant emotionally deep experiences, to be possible in future works that wish to research what could be the emotion-related part of our system (more on this topic, again, on the future work section of this document).

But we remind we only focused on top-down isometric viewed TPS games, and also did not delve into producing other genre-specific visual experiences.

Lastly, Battle Cam and its "visual attention" approach is notably interesting, and should be taken further note off for future work based off ours, that is specializing on RTS games.

2.4 Intelligent Camera Control in a Virtual Environment

Intelligent Camera Control in a Virtual Environment[SD97] specified a very similar system, but only aimed for exploring virtual worlds.

Their virtual world was a museum, and they wanted to provide tour guides the best possible system to assist them with touring the museum.

Thus, their goal was to provide an easier interface than direct control; the user would select an exhibit and the camera, knowing how to traverse the world as well as how to take a shot of the exhibit, whatever it may be, would do just that.

Their goal was to separate user interface from the underlying framework that handles camera motioning and positioning, by providing a good intelligent camera.

Our work differs in the way that our virtual world is dynamic, and its state is very volatile. It is a video-game match, constantly changing.

More than that, we wished to have an automatic control over what the camera is going to show rather than, for instance, having a video-game expert pointing out what he wants the camera to show the audience, and the camera just moving around based on his choices. Our wish is for the camera to know how to choose, and choose.

Traditionally, as demonstrated in this work, one had an approach where the viewer interacted with, let us call it a Shot Selector. With it, the viewer selected a shot, an object to view, and then the camera system would navigate the camera, as necessary, to show it.

Then, works in intelligent camera began working that interaction to be as simple as possible, even to the point of eliminating the interaction between the viewer and the process of selecting a shot, by developing an A.I. that emulates the intelligence, and notion of a scene, of a movie director, to automatically provide a cinematic visualization of the virtual world.

What is important is that the viewer, the spectator, does not interact with the Shot Selector. Instead, the Shot Selector is in constant communication with another module, one that can read the world and present to the Shot Selector how important all the possible points of interest are, so it can select one to shoot.

This work is a good analogy to ours, but fundamentally different. In our work, the system has, within itself, the ability to query the virtual world about what is important to show the viewer, instead of having someone doing that (like the tour guide).

Next we will look into one more section of related work: that developed in the domain of sports broadcasting.

2.5 Sports Broadcasting

In the domain of studies in Sports Broadcasting, we focused on studies done on highlights extraction from sports videos. Though the goal of the studies are usually to use low-level audio-visual data, like sound/color/shape/motion, to recognize if a highlighted event is going on (a soccer goal, for instance), some of them developed some ground work that we feel is relevant to take into consideration. After all, we believe it is important to probe

what studies have been done to conventional sports, since it is most likely analogous to our own effort to recognize highlighted events (the match-progress cause-effect events to show the spectator).

Let us take into consideration two works: "High-level Event Detection in Broadcast Sports Video"[Rea05] by Niall Rea, and "An Overview of multi-modal techniques"[ALM03] by Adami, Leonardi, and Migliorati. They explained that a lot of the work done on recognizing match-related semantic events in sports video focusesed on recognizing the type of shots shown at particular times, in what succession they appear, and from that, depending on the sport, they applied heuristics, that are observable from matches, to infer what type of event occurred. So, paraphrasing, *they applied shot classification, and used observable heuristics to interpret it.*

Rea observed and applied this shot classification to snooker and tennis, and Adami, Leonardi and Migliorati offered an overview over techniques developed for soccer, baseball, cricket and basketball.

Continuing on, we quote another work: "Algorithms and system for segmentation and structure analysis in soccer video"[XXC+01]:

there are some production rules that sports video-makers usually follow[BS96]. Producers typically aim to:

- Convey the global status of the game.

- Closely follow actions in the field. In order to meet these objectives, we have observed in soccer games that:

- During the play, it mostly stays in global view to convey the whole status of the game; interrupted by short zoom-ins or close-ups to follow up the players' action.

- During the break, zoom-ins and close-ups tends to be the majority as they can effectively show the cause and effect of the break (such as why a foul would happen, its consequence, etc.).

We felt it was important to quote the whole segment so we could note it stated that shot-classification is sport-dependant, which is to say, that camera work is not the same for soccer, tennis, American football and so on, but also to point out the aim of broadcasting producers, which are to "Convey the global status of the game. Closely follow actions in the field."

We believe that these many studies demonstrated that different kinds of sports are approached in different ways by broadcasting crews, but that within the sport, the approach is always the same. That is the basis on top of which shot-classification functions. We think that is because there are heuristics to adhere too, and even though they *are the same*, they do not translate in the same way across sports: for example, global view in soccer is a wide side-view[XXC+01] [ALM03], while a global view in Tennis is a wide diagonal view, behind a player[Rea05] [ALM03] but both will use global views when there are no interesting actions to closely follow and show the viewer, whic is to say, when the game-independant heuristics apply.

This type of reasoning was employed in our solution.

It was important for us what broadcasting crews prioritize, at a particular state of the match. And all these works left ground work validating our approach towards analysing

and researching just that. Knowing, confirmed by literature above, that one can relate shots to what the crew wishes to show the viewer (global view shows the state of the game, for instance), we can extract what is a general priority list for a broadcasting crew in general, into heuristics we can use for our system, simply by extensive and careful observation.

We will cover this in the next chapter of this document. How we did this, and what results it actually produced.

Chapter 3

Heuristic Research

AutoBroad is a vision of a spectating system that will allow spectators to watch a video-game match with the same or superior quality they would watch a television broadcast of a conventional sport. It would employ several key features, most of which have never been seen or concepted before, for this particular domain of Electronic Sports:

- Real-time game-state awareness.
- Automatic and correct event-priority judgement.
- Automatic broadcasting-heuristics-based decision making.
- As game-independent as possible.
- Light and uncombersome to the video-game.
- Relative global and individual information feedback.
- Instant-replay.
- Multiple viewports for more than one focus.
- Animations to transition between more than one focus.
- Cinematic or cinematographic presentation.

Certain features certainly involve other domains of research, and could be considered well studied in past works. This is why we decided to focus on the first six features, leaving the rest, which we feel to be secondary, to be developed and explored in future work.

It is a relevant and important first step into building a system which we feel that, in its optimal state, will further the overall investigative field of video-games and be of use to both the gaming industry as well as the sports industry.

In this chapter, we will present the research we did in order to design and implement our system, and then we will present the design and implementation of the system itself.

3.1 Preliminary Research

In order to build our system, we needed two basic functionalities which have found to be, as of yet, unexplored by the scientific community: interpret the game, and use that information to make directing decisions over what to shoot.

The system has to be able to reflect both knowledge of the game-match and the wisdom to know what to show, so our first step was to perform careful observation of sports and e-sports in order to extract from them information about what broadcasting teams prioritize to show, on a given state of a match.

We neglected the real complexities behind sports broadcasting production, though, as this largely focuses on producing a determined emotional impact on a viewer, which as we have said, will not be our focus. We just looked for decisions that one can derive from extensive observation, and employ into a successful system of heuristics for our Game-State Interpreter and Shot Manager to use.

For this purpose, we observed 56 best-of-three matches from the Starcraft 2[16] championship GSL[6], season 2, held throughout October and November, 2010. We observed about 4 matches of League of Legends[14], from the WCG tournament[8], 2010. We also observed all world finals in the ESL 4[7], 2010, for WoW Arena[20] , Counter Strike 1.6[18] and Quake Live[5]. The analysis of WoW Arena and Counter Strike 1.6 was assisted by an expert on the games who, at different times, was on the top 15-20 rank in his realm for WoW PVP, and was a competitive gamer of Counter Strike 1.6. His help was invaluable to discern what went on in the observed matches of these two very fast-paced games, and what was focused by the team of broadcasters, since we were not that familiar with those two games and, as we have made sure to point out, if the viewer is not familiar with the games, it is difficult to understand what is taking place, and thus difficult to derive entertainment from the broadcast.

We have also observed 3 soccer matches, 2 Moto Grand Prix races, 1 Formula 1 race, and 1 American Football match, and took into account the studies of the works discussed previously (chapter 2.5), which included baseball, soccer, cricket, snooker and tennis.

Again, we remind the reader there is a quick list of the games and game genres we will be discussing, displayed at the introductory chapter(section 1.1), as well as in the appendix, sections 6.3 through 6.6. All in case the reader is ever unfamiliar with any of these games.

First we will discuss our conclusions on Conventional Sports broadcasting, secondly we will discuss our conclusions on E-Sports broadcasting. Thirdly, we will offer general concluding remarks, in which we will define our final heuristics.

Our final aim is to describe heuristics employed by current day broadcasting teams, towards shot/event priority.

3.1.1 Sports

Racing sports present the similar challenge of filming concurrent states and scenes. Interesting things may be going on at several places at the same time, or simply unpredictably.

It is impossible for the broadcasting crew to guess when a car or a bike is going to crash, and thus they easily miss it.

Techniques that are clearly used to deal with these issues are focused on anticipating relevant events before they happen; they prioritize clusters of opponents above all, for that is most likely to produce a crash, which apart from the final cross of the line, is a priority to the viewer. If none are available, then racers in the lead are prioritized. If racers in the lead are clustered, then leaving the shot is very risky due to the very high priority of the shot which is conflict for winning position. They normally keep up with them, only changing perspectives at best.

What is done when no action is taking place is covered at the end.

Team sports like soccer and American football present the challenge of catching tactical details that happen before players get near the center of the action (the ball). Everything that happens near the ball is usually easily caught, and all the relevant action does happen there. But, for example, in American football, it is common for a runner to come out of nowhere, and suddenly receive a pass. It would be interesting to see that building up, the whole process of anticipation into conclusion.

Some techniques applied are varyingly dynamic Wide-shots (global shots). Which is to say, for no apparent reason, we see tower shots or wide-side shots of a very wide area, to catch a large number of players. Detail and first-hand experience is exchanged for tactical awareness, so that if the player starts dashing, everyone sees it, the zoom can be made, and a glorious moment observed live. Then there are those moments where nothing happens, discussed next.

A general issue that sports broadcasting seems to have is the existence of uninteresting moments, and of moments the broadcasting crew misses. Techniques to deal with this include commentary, a very important technique to keep the viewer interested, and instant-replay, which kills both birds with one stone.

Instant-replay fills in for the uninteresting moments, and can also cover for missed moments, or even caught moments from a better and more exciting perspective (for example, to show when the American football player realized he had a chance to pierce through the defensive line). Missed moments are caught by crew assistants and cameras on site that filmed the event, even though they were not streaming back to the viewer.

Meanwhile, commentators are very important when nothing is happening; they choose varying global views, or close ups to players, audience or the stands, and talk about the match, or race: about the participants, their history, history of the championship, strategies, and so on. They are also a great deal responsible for the emotional impact that the broadcasting has on the viewer, as we have already discussed (section 1.3).

So, in conclusion, it is all about the action: how to catch it, how to show it. When there is little chance of action, when there is no conflict, then showing/discussing strategy and/or tactics seems to be the norm.

Something important we also observed was that broadcasting teams never stay in one shot for too long. Even if they are tracking the same thing, they will change shots as soon as they can, if enough time has gone by; but this is action-dependant. One can never, ever, lose the action[BS96].

3.1.2 E-Sports

The problems are very general, and not at all genre-dependant.

There is a challenge of filming concurrent events, as well as of catching all relevant tactical details. Also, especially in RTS and TPS games, there seemed to be a large amount of uninteresting spans of time to deal with.

Techniques currently applied seek to solve these issues by simply employing expert and professional players of the E-Sport. They control the camera, thus supposedly catching all relevant tactical details in a satisfactory manner, as well as providing commentary to fill in the void of interesting events. Good commentators are, again, crucial.

There is no application of instant-replay during matches. Though to be fair, there are games where it would be impossible to do so during rounds (Counter Strike, Quake and WoW Arena, to mention a few).

Something that is done to enhance viewer experience depends largely on the game, and what spectating features it offers.

In Counter Strike 1.6, broadcasters can alternate between player-cams, and a spectator camera that sits high above the map, so they can see the players, colored blue and red according to their team, move around the map. This camera gives them a tactical view over everything, and they largely use that to talk about their strategy regarding positioning, and predict future tactics, in the eventless quietness of the start of the match, until opposing players first meet.

In WoW Arena, they only have player cameras and they choose the player according to the state of the battle, and his class. For example, one damage dealer trying to kill a healer is most probably not going to work, so they will be following the crowd controller and the other damage dealer, to see if they can neutralize the opposing healer and crowd controller duo.

In Starcraft 2, as well as in all RTS games and TPS games that share the same kind of camera control, which is simply panning and limited zooming, broadcasters have their own camera. They are a third player, that simply does not have any faction, and can see everything. They largely use a feature that shows what is the visibility of a player, to cause expectation and tension towards the spring of a trap, or the discovery of a strategy that would still take 5 minutes to reach fruition.

In fact, this ability to show the spectator what a certain player is viewing sets E-Sports apart from most conventional sports. An analogy we can offer is that of poker(and other like-minded card games); in it, broadcasters can also show the viewer what cards the players hold. The tension is higher when the viewer knows another player should fold, and sees him trying to bluff someone who is not falling for the bluff. A broadcasting team can instill that experience almost to any video-game, since the control is much bigger,

and genres FPS, TPS and RTS have a strong basis on opponent visibility; scouting the opponent, knowing simply where he is and what he is doing, is very important in every match we observed. This sets E-sports broadcasting apart from conventional sports broadcasting, in terms of broadcasting, since there is that one feature, which is present and can be manipulated in every high-competition e-sport, at the exception of Beat 'Em Ups.

In terms of genre-independent techniques, in every e-sport that we observed there is a tendency to show the development of a fight, from the point of view of the survivor, which is to say, of the losing side. We believe that is because it gives a much more dramatic perspective, since by all accounts, the losing side should lose, but still always has a chance to survive, and even win. This happens remarkably more often than one would expect.

And also, like-minded with conventional sports, uninteresting moments are filled by commentators who discuss strategies, history of the players, championship, game-map, etc. Commentators seem to be as important in electronic sports as they are in conventional sports, having exactly the same duties and responsibilities.

Additionaly, we observed that broadcasting crews noticed unexpected events from game-state related signals. In Starcraft 2, they see in a video feed of the build order that a weird building has been constructed, and only then do they move to show it. In League of Legends, an item purchase can go unnoticed until the point that character deals a great amount of unexpected damage; by observing how fast the health meter of the opposing character depletes, the broadcaster will then observe and point out the purchase of said item. It is easy to miss interesting events, and broadcasting teams are warned by observing indirect game-state feeds, their personal attention allowing. Though again, there are no cameras recording the event off the live feed to later show it through instant-replay.

And in conclusion, e-sports broadcasting shares the same basic interests, and follows the same basic principles that we concluded in the previous section.

We will note, though, that an e-sports broadcasting system does not have the backup of an instant-replay system, nor a big number of cameras set up to take different kinds of shots. FPS and TPA games have global shots(sometimes) and player cameras, and RTS games only have movable global shots, some of them providing player view, but not player camera which means no one ever really knows what the player is seeing.

3.2 Concluding Remarks

In this section, we will define what we have observed into a terminology that we will use in the course of the description of our solution.

First, we have observed that action is related to *conflict*. When there is conflict between two or more players, especially over a given objective (ball, position in race, power-up, special resource, etc), then one must focus that.

When two opponent soccer players get near each other, there is a chance of conflict; if the ball is near them, even more. When a baseball batter hits the ball and runs off, as he approaches each base, the conflict arises, but there is also conflict arising from the ball getting near a catcher. Concurrent conflicts, yes, but one is more important than the other. When players in Quake Live approach each other, conflict arises, and if they approach each other while approaching a power-up, the conflict is more important. The notion of conflict is game-dependent, but we feel it defines the high probability of action taking place, presently or in the future, and we will use that word to define that as an heuristic that broadcasting teams give importance too: *conflict*, which most likely translates to action taking place presently, or soon into the future.

Then, there is the *unexpected*. This is caught by some unexpected change in the game-state progress. For example, in Formula 1, when a car crashes, the driver quickly drops from his position to last in a few seconds. In Starcraft 2, if a sneak attack is carried out against a base, a lot of workers start to suddenly die. Once a missed moment is noticed, it takes a high priority over anything that is taking place and is not real-time conflict. The aftermath, or ongoing event, is focused on and later, if possible, a replay is shown.

Then, there is decision, which is to say, *tactical/strategical awareness*. When none of the former are likely happening, we observe the broadcasting crew just alternating between shots, showing global view or partial that, to show what strategies/tactics are being employed. In Starcraft 2, they show the buildings, and units being created. In League of Legends, they show what vision wards have been placed, how the characters are positioning or moving around the map. In Quake, they discuss the power-ups that are respawning, and what advantage they will provide either of the player, who is closer, and so on.

Finally, *when all else fails*, when all strategy/tactics have been showed, and none of the other two circumstances are likely, they just alternate between close ups of players, audience, or player cameras, or they focus on the winning player, or iterate through other players, or show replays of the most recently amazing events, and so forth. They just iterate between these alternative shots, discussing everything they can to keep viewers interested. This is where commentators need to shine.

Generally, there is also an "interest" rate. Which is to say, if there is more than one conflict going on, choose the conflict that involves the player that is doing better, because it is more interesting to see what happens to that player. Then, inside that exchange, the survivor (the one losing), is the one broadcasting crews need to follow. We observed though that even in these circumstances, shot changes should occur, so if the conflict drags out, showing it from other perspectives should be done.

These are the heuristics we can conclude from our observation: *Conflict, Unexpected,*

Tactical/Strategical Awareness, and "When-All-Else-Fails".

Plus, there is the "interest" rate, which rates higher ranking players as more interesting, and the "time" rate, which rates how much time on a shot is *too much time on a shot.*

We believe these definitions to be sufficient for our match-progress event classification and prioritization.

But we remind that there is more to the best possible broadcasting system: emotion-related heuristics and production knowledge, to help relay a scene in a way it will better influence the emotions of viewers and, thus, entertainment; and commentator. We focused our research and development on a system that could show events in such a way a viewer will understand the cause-effect relationship between them, and be able to discern what is going on, and why. Automatically, this means showing a viewer the action, and overall, what a general viewer would think interesting to see.

So our system could make good use of a commentator, who would watch our system function, and commentate as the match went on. But the emotion-related module was not in our group of focused features, and so we did not consider it during our research.

With the system of heuristics defined, a way to better judge what to do when looking at the state of a match, we were able to proceed with the definition of a real-time automatic broadcasting system for multiplayer video-game matches.

Chapter 4

Architecture

The final objective is to have an automatic real-time broadcasting of a certain game. For that purpose, the system has to be built taking into account the development platform, but mostly, the game in question.

But before that stage, we first designed a model that can be used as the basis for any architectural model of an automatic real-time broadcasting system. We will start by defining that model architecture before we look at the actual implemented architecture, which took into account the development platform and the video-game in question.

4.1 Model Architecture

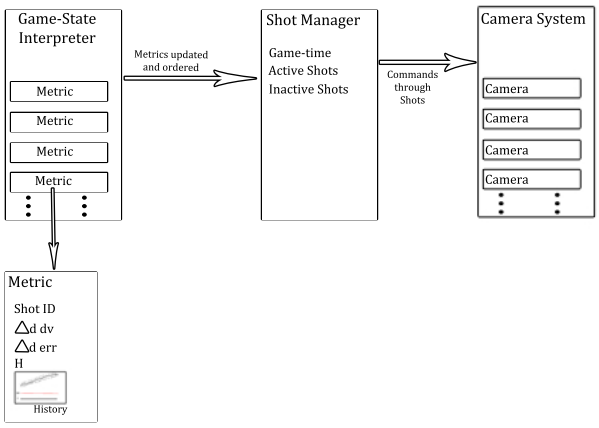


Figure 4.1: Model of Base Architecture

Figure 4.1 shows the base architecture that we are proposing should be used whenever a system of this kind is developed.

It has the three general modules we have mentioned before: the Game-State Interpreter, the Shot Manager and the Camera System. Additionally, it includes the concept

of “Shot” and “Metric”

4.1.1 Metric

A metric is a parameter of the game that has a priority value. It must be one-dimensional, low level, and quick to measure: position, resources, health, energy, ammo, ability was used, etc; any low-level information that is related to a location in the game, from which we can derive interest, is a metric.

The only thing these metrics are aware of is how to calculate their priority value, and to what shot they are related too, as we can see in the figure (ShotID). The idea behind this is every metric is associated with a game object, like a character, a squad, a building, something that has a position to take a shot off; it is an easy shortcut to knowing how the metrics relate with the broadcasting, what exactly are they evaluating the priority of, in terms of game location. The ending goal is to know what to shoot, as in, what to look at, which is the same thing as asking which “shot” to show.

Then it has a distance to the desired value (dv). The desired value is the value at which point the metric attains maximum priority. So the fluctuation of the base priority of the metric is directly proportionate to the distance to the desired value.

Then there is a distance to the error margin (err), which along with “history”, allows the metric to monitor how it evolves during the match, and to make extra accounts should things go unexpected. History predicts what the next value will be, the “projected value”, and then when the metric gets around to actually calculating it in the present, it compares the actual value with the projected value, and if they differ by more than the error margin, this is something unexpected that the Metric takes into account when calculating its final priority value.

To summarize, the Metric will calculate its priority value following two approaches:

In a bottom-up approach, there is a relationship between the abruptness of a spike and the level of interest. An unexpected event almost always carries a high level of interest in broadcasting, which means the acuter the spike, the higher the interest. Thus, the farther the graph falls under, or rises above, the error margin, the more it will contribute to the overall priority value.

Then, in a top-down approach, there is a relationship between how close the graph is to the desired value, which is pre-determined as a point of interest. The shorter the distance between the actual value and the desired value, the more it will contribute to the overall priority value.

Finally, there is the heuristic value, which makes a final contribution to calculating the final priority value of a metric.

Heuristics

Depending on how one builds the metrics, they can switch between heuristics, but we believe that, at any time, they should only have one heuristic value.

Reminding, our proposed heuristics were chosen and built out of observing matches in world-wide competitions of the following E-Sports: WoW arena, Counter Strike 1.6, Quake Live, Starcraft 2 and League of Legends. And also from data collected on Conventional Sports, of which we emphasize Moto Grand Prix, Formula 1, Soccer and American football. Having that in consideration, we defined them as so:

First, *prioritize conflict*. Every interaction between opponents is more interesting than watching other things happening in a match. Even in Starcraft 2, it is better to see what the scout is doing in the opponent's base, what he is scouting, than hanging back watching a building being built. If there is more than one conflict, prioritize the one involving the overall winning player, which is the most important conflict. But then, having picked a conflict, do so from the point of view of the loser, which is to say the player in a disadvantage. This follows the heuristic to play the survivor effect.

Next we *prioritize unexpected*. It is most likely something that was lost and was not shown to the spectator. The aftermath must be shown, and the timing of what was lost marked for later replay. If the interpreter works so well it pointed it out before it happened, and so it was shown, all the better. As an example, in League of Legends, a hero can suddenly activate a combo of abilities, using items, and make a very important push while a fight is being focused, one that actually dissolves into an uninteresting stalemate by the time the push has resulted in dead towers. In Counter Strike 1.6, the system might be focusing on a shoot-out which results in no kills, and then suddenly a player kills three players under 5 seconds, which is a common occurrence in this game.

Next, we *prioritize tactically relevant actions*. Builds, buying items, positioning, etc. They are the wide-shots or localized shots whose use is merely to show the spectators a tactical detail that might be relevant for the outcome of the eventual conflict, to show them what the player is thinking and planning. For example, in Starcraft 2, the builds the players are doing have to be shown, though the system does not dwindle and wait for a building to finish construction. What units the player is making need to be seen, but the system also does not dwindle next to the production building as it produces more and more of the same unit. We see what building it is, and that is sufficient. We see what unit, or what upgrade the player is purchasing, and that is sufficient.

Likewise, Counter Strike 1.6 only needs an initial and very ephemeral view of how the players are positioning themselves, because they are just going to kill each other afterwards. In League of Legends, we see a player buying items but we do not wish to sit around focused on him when he is not doing anything. Likewise for other games, tactically relevant actions are relevant and important, and the system must make sure to give them the respective importance, but we never want to stay on the location of these events which is why they are the last valued dynamic priority.

The last priority, static, is the *"when-all-else-fails" priority*. There are a number of directing alternatives to apply: Focus on winning player, the one with the best overall

performance; replay some event, ordered by pair (most recent, number of times shown); iterate through other players; show a complete overview of the game-world; and so on. For this reason, and for the "survivor effect" as well as for the interest rate, there should always be, and will be in our implementation, metrics that represent player efficiency, so the system is always aware of who is ahead, and how far ahead they are.

4.1.2 Game-State Interpreter

The Game State Interpreter manages the metrics. It marks the pace at which they update, and keeps them ordered.

As mentioned before, we are realizing situation assessment. To make it easier on nomenclature, we decided to coin the term "Game-State Interpreting", as a name for the method in which we first derive high-level information from low-level game-state information (situation assessment in an ongoing video-game match), and then interpret that high-level information into a specific language or ontology.

Our Game-State Interpreter will examine game-state (the situation), understand it, and then interpret it into broadcasting language; this means defining points of interest in the game according to broadcasting heuristics.

4.1.3 Shot Manager

Our approach is to employ a low cost but effective situational assessment of actions taking part in the game. This means using one-dimensional metrics from which the system can read and understand game-state progress without affecting performance. So far, this is already done by the Game-State Interpreter.

The Shot Manager is aware of the broadcasting progress, what shots have been active, which have been inactive, and for how long. It has the job of overall management of the whole system.

Bearing that in consideration, the module adds a deciding factor to the heuristics management, which is *time*, aiming to follow the last heuristic: *it is not ideal to stay on the same shot for too much time.* So, depending on the shot, it will decide if, despite of it having a higher priority, it should change shots anyways. But whatever time count he uses as a threshold needs to take into consideration what kind of priority values it is dealing with. For example, we want to film the conflict, but we have been following the winning player-cam for 20 seconds, so maybe it is time to show the losing player-cam. But only if the winning player is not in the middle of a conflict.

The module also controls and applies the *interest and survivor* effects. Though it uses the metrics to know which player is ahead, it is the Shot Manager who applies these heuristics.

So all in all, we have six major guidelines based on observable heuristics, four of which are already defined by the Game-State Interpreter; six guidelines that will hopefully allow the system to be as dynamic as a real broadcasting system, without the emotion-related behaviour.

Having chosen what shot to show, the Shot Manager is ready to contact the camera system with orders for the appropriate cameras, by passing on to it which shots are active, as well as any information about them that might be new.

Finally, the Shot Manager also needs to recognize parallel or concurrent events. If it has to pick between two, or more, sufficiently approximate priorities, it means it has missed one. If this is so, it should save the time of the moment, so it can issue an instant-replay command on the camera that recorded the missed event, at the appropriate time.

4.1.4 Camera System

Unlike the previous sections, we have barely developed on this module.

In actual fact, our camera system is basically just one camera; it switches between "shots", which are basically encapsulated information about camera configuration that we also used for extra purposes (explained in the next section). Strictly speaking, the camera system is as complex as strictly necessary to test the other modules. We did this because we sought to test the other modules and not this extensively researched and developed module, but mainly because the development platform did not support dynamic viewport management, which is the basis of what we envisioned for this module.

Our greatest concern with camera work in video-game broadcasting was, in fact, transitions. Navigating the camera between points of interest is sometimes done so quickly that the spectator is confused, or too slowly to catch an event; we covered this issue in section 1.2. To solve it, we envisioned a system where cameras, or shots, never transitioned.

That is not to say they never moved, since they do need to follow their moving targets (like characters), but the idea is that the camera system would use *viewport animations* as a way of transitioning between camera feeds.

The idea is that the camera feeds are always fixed on their targets, and when their targets are deemed important, they simply transition into the screen as a viewport. This would also enable having more than one camera visible at any time, making it easier to capture parallel events live, giving the spectator the choice to focus on whichever he prefers.

When transitioning between camera feeds, the transition would then be smooth, to maintain frame-rate coherence, so not to lose the spectator, and also to give a futuristic edgy look. A look that is not unique, as it has been seen in television series like 24[2] and animated series, or even on movies like Hulk[1] and Scott Pilgrim vs the world[15].

We believe this technique would utterly simplify camera navigation to its raw, simple, and long mastered form, all the while adding viewer experience. It also makes it easier for the system to catch the right shots, as it will normally have two or more different viewports being shown at any given time.

These would be managed by the Shot Manager, since it knows which are more important, how long they have been present, etc.

Again, the cameras themselves, the feeds, would be statically associated to points of interest and, at most, animate themselves to film said points of interest in other ways.

4.2 Implementing AutoBroad on a TPS Game

Now that we have looked at the base architecture for a general automatic real-time broadcasting application for video-games, we can apply it to our domain of testing.

We implemented the system on an indie video-game developed in Unreal Development Kit(UDK) [3], named Shadow Conclave [19]. It featured one on one matches only. To better understand how we described and defined the metrics system, we feel it is important to understand the game itself.

4.2.1 Shadow Conclave, a TPS video-game

UDK[3], as the name states, is the development kit to develop games for Unreal Engine 3.

Shadow Conclave is a third person strategy game similar to League of Legends, Defence of The Ancients and Heroes of Newerth, though different in many aspects.

First off, it is not team based, and the players do not actually fight each other; each player picks a unique character with a unique set of skills, and plays for a pre-determined and fixed amount of time, all against all. It is his objective to steal as much money as he can from the city where the match takes place. He does that by breaking into houses and stealing from treasure chests.

There are militia patrolling though, and if they catch the player he will lose time in jail, and money. There are three kinds of Militia: Standard Militia, Rats(they hear and see farther), and Pro Militia (they cost the player more money).

Also, there is a fourth type of NPC, called “Tax Messenger”. They show up at random times, and if a player catches them, he will steal their “taxes”, gaining money/points.

Thieves can interact directly by causing each other to be stunned, or disoriented. There are two thieves in the version of the game on which we tested: Circus Freak and Hunter, which we can see in figure 4.2

Figure 4.2: Characters of Shadow Conclave

Circus Freak is faster than the Hunter and has two abilities he can use: ”Fear” causes all near him to become disoriented (move around randomly), ”Break an Entering” opens a door almost immediately, but makes noise. Hunter is quieter than the Freak and has

two abilities she can use: "Stun" causes a particular foe to stop in its tracks for 5 seconds and "Value Sensing" causes valuables to glow through the fog of visibility.

All the thieves do is go into houses and steal money from treasure chests. At the end of each match, there is a "raid" event, where each player must make his way to an exit point in order to flee the city, and not suffer consequences.

The game is played with the top-down isometric view that is the brand of RTS and TPS games, using the keyboard to move the character around though, instead of mouse clicking. Mouse clicking aids the use of abilities and enables interaction with doors and treasure chests.

Now that we have a clear picture of what the game involves, we will delve into our implemented architecture.

It is important to note that the developers of this video-game numbered at four and all aided in the defining of the metrics, as well as with testing and optimizing the implementation of AutoBroad.

AutoBroad went through a series of preliminary tests to optimize the parameters that define its mathematical calculations and overall valuation of the points of interest of the game. This testing was done over matches played by these developers, experts on the game.

4.2.2 Realizing AutoBroad

The fact this game is "player-based", which is to say, any action is happening around the *one avatar/character* that the player is controlling, was decisive in some of our decisions on how to implement, particularly, the interaction between the Game-State Interpreter and the Shot Manager. All other decisions are mainly due to the abilities, or limitations, of the development platform UDK.

The simplified UML class model of the implemented AutoBroad system is shown in figure 4.3. The most surprising difference between it and the base architecture is the presence and size of the module "Shot", which should only be encapsulated information about camera configuration.

Additionally, a class "Actor" also features in it, because it is the highest level of object in unreal script, directly below "Object", so we will be referring to these "modules", as well as the characters used by the players, and even the houses and treasures and anything else, as actors.

Top Level Behaviour

Initially, all the behaviour in "Shot" would be in the camera actor, while information that dictates it, yes, would be encapsulated in the Shot actor, as we showed in previous sections. That was the plan, but due to some limitations inherent to UDK, which do not warrant an explanation, it was easier and cleaner to turn what should only be information encapsulation into a whole proxy which would dictate camera behaviour.

Since this is a player-based game, we made every heuristic directly related to each character/thief. Likewise, we have two shots, one for each player, always ready to be

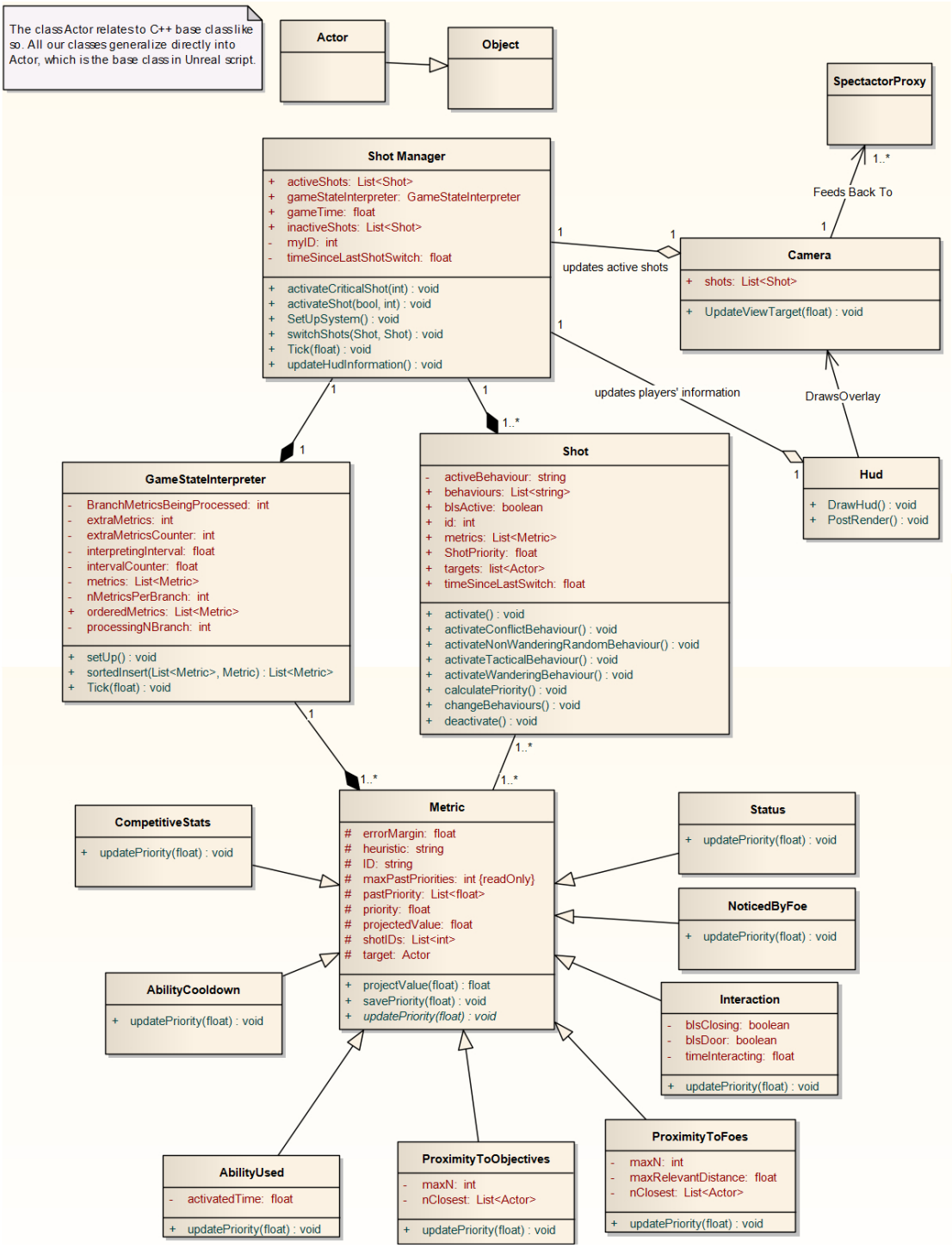


Figure 4.3: UML Class Model of our implemented AutoBroad

picked to show what they, as proxies of the camera, are filming. For this reason, we felt it was more efficient to insert that concept into the functions of the Shot Manager, so instead of processing the metrics directly, the Shot Manager processes Shots.

The Shot already has a list of all metrics that concern it, and a priority value which tells the Shot Manager how important it is. The math behind all of this will be in the next section.

The metrics are updated by the Game-State interpreter, and the priority value of the Shot is updated according to the command of Shot Manager, and then the Shot Manager looks at the Shots, picks out which one it wants to show, and passes it over to the camera. The camera will then use the information held by the respective Shot to stream its feed.

The position depends on the targets of the Shot, and on the "behaviour" value. A shot will usually have only one target, the player, but it can have more(player plus NPCs, or treasures), in which case the camera positions itself on the "center of mass" between all targets. Behaviour dictates Zoom behaviour, and it currently has three values: "Conflict", "Tactical" and "Wandering". "Conflict" instructs the camera to hover around the closer zoom value, "Tactical" instructs the camera to hover around the farthest zoom value, both values static, and "Wandering" instructs the camera to wander around the map. This last one we have not implemented.

Game-State Interpreter works exactly as described previously. Now let us look at the metrics, the more game-specific element of the system.

Bottom Level Behaviour

We defined 8 metrics for Shadow Conclave.

- *Competitive Stats* - This metric informs the Shot Manager of who is winning.
- *Ability Cooldown* - This metric informs the Shot Manager that this character is about to gain one more avenue of action.
- *Ability Used* - This metric informs the Shot Manager that an ability has been used.
- *Proximity to Objectives* - This informs the Shot Manager about how close a given player is to treasure chests or raid exit points.
- *Proximity to Foes* - This metric informs the Shot Manager about how close a given player is to Standard Militia, Rats, Pro Militia, Tax Messengers and/or, more importantly, other players.

- *Interaction* - This metric informs the Shot Manager of whether the player is interacting with either a door or a treasure chest, and whether he is opening or closing, and how far he is to completing the interaction.

- *Noticed By Foe* - This metric informs the Shot Manager about whether or not the player was seen or heard, whether he is being pursued, and if yes, by either Standard Militia, Rats, Pro Militia or Tax Messenger. It also informs it of how much time has passed since the player began being pursued.

- *Status* - This metric informs the Shot Manager of whether a player has been stunned, disoriented or arrested, and of how long until he comes out and resumes normal status.

In the next section, we further discuss them.

Concluding

And this is the architecture we used. The way it functions is that the Game-State Interpreter makes sure that within every decision cycle of the Shot Manager (half a

Figure 4.4: Screenshot of Hud

second), all metrics are updated. It does that by updating as few as it can in each tick. We have a total of 8 metrics which we believe cover all approaches towards measuring points of interest, for the two characters, which makes for a total of 16.

Every half a second, the Shot Manager makes its analysis, telling each Shot to calculate its own priority value so it can judge if it should perform a switch. At the same time, it keeps the HUD informed of all information it needs: the abilities of the player as well as the gold that they hold plus the overall game time plus a minimap like the one players have access too. We can see this in figure 4.4.

The game features one on one matches and is simplistic enough to make testing easy, but complex enough to make it worthwhile.

As to the inner workings of the system itself, we will discuss it further in the next section.

4.3 Implementing AutoBroad on Shadow Conclave

In this chapter, we will cover the inner workings of every module. From the math behind the calculations of priority values of the modules "Shot" and "Metric", to the timings and control the Shot Manager performs, to how exactly the Game-State Interpreter manages the metrics so they all update within a fixed amount of time while keeping processing as light as possible.

4.3.1 Metrics

There are a total of 8 metrics. They are related to players, and there are two of them, so that makes them 16.

Each metric follows the same reasoning. Their priority value always fluctuates between 0 to 100. To calculate their priority, they look at the value of the parameter they are tracing and apply a formula that puts that value between 0 and 100.

The other modules will take it from there. One of our thoughts was to apply a general formula of growth through way of derivatives and parametric curves. This would work well with metrics like Proximity and Ability Cooldown, a study of how the metric evolved, deriving from that its tendency and then marking anything diving and sky-rocketing out of the tendency as more important, all the while maintaining the top-down approach of evaluating the straight distance between actual value and desired value. This would be a solution more keen to the one described in the base model, and might be applicable to some games, but not to this one.

Metrics like Ability used and Competitive Stats do not exactly fluctuate. They make hard shifts between values (on -¿ off, 80-¿ 20, etc) which makes it difficult to apply a parametric function that makes sense. Also, there is the issue of making sure all the metrics are balanced. At what distance is the proximity metric as important as the Ability Cooldown metric, when it is at 2 seconds from finishing? Having an upper and bottom limit, a desired value for when the priority value of the metric is at 100 and a negligible value for when it is at 0, makes it possible for all metrics to be balanced.

They fluctuate between 0 and 100, thus being equalized, while internally, they value themselves in very different ways.

What we have in place to portray the behaviour behind history and error margin, as discussed in the base model, is a fixed error margin and projecting formula.

Every metric starts with a thirty percent error margin for projected values. They project a value thirty percent over the current actual value, if the priority value has been rising, and thirty percent below the current actual value, if the priority value has been declining.

So if the current value is 40, the projected value is calculated to 52. When the next update arrives, the actual value will be compared against the projected value, and if it is 15.6 more, or less, than 52, then the priority value of the metric will receive a modification through the following conditional formula:

If the actual value is over the projected value by more than the error margin, then the new priority value will be:

$$priorityvalue = priorityvalue + \frac{-projectedvalue + newpriority}{newpriority} - errormargin \quad (4.1)$$

If the actual value is below the projected value by more than the error margin, then the new priority value will be:

$$priorityvalue = priorityvalue + \frac{projectedvalue - newpriority}{newpriority} + errormargin \quad (4.2)$$

For example, if the actual value was 80, for the projected value 52, while the error margin was still the standard 30, then we would have:

$$priorityvalue = 80 + \frac{80 - 52}{80} - 30 \quad (4.3)$$

Additionally, every time the metric priority value misses the projected value by more than the error margin, we count that as an error. At each 10 errors, we adjust the error margin to the same thing we add to the priority value every time it experiences that unexpected development:

$$errormargin = \frac{priorityvalue - projectvalue}{priorityvalue} \quad (4.4)$$

Or, as with above, the inverse formula.

$$errormargin = \frac{priorityvalue - projectvalue}{priorityvalue} \quad (4.5)$$

But we also established a fixed number for the error margin, beyond which we will not adjust. Sixty percent seemed to work fine in our tests.

Overall, early tests showed this working admissibly well. The metrics that spike more have a broader error margin, the ones that spike less have as little as the standard error margin.

And now we will describe each metric individually.

Competitive Stats

This metric monitors how well the player is doing, by way of three variables: Points, Time since arrested, and number of consequent houses the player has robbed without being caught. Each contributes to a part of the overall priority value. Let us call them, respectively, t1, t2 and t3.

Heuristic value is fixed at *"tactical"*.

- *Points t1* - The formula to calculate this portion of the overall priority value depends only on the maximum number of chests, let us call it maxchests, and on how many points the player has, let us call it currentpoints. It goes as follows:

$$t1 = \frac{currentpoints * 100}{maxchests * 100} \tag{4.6}$$

  We multiply maxchests by 100 because the gold treasure chests hold is randomly calculated to be between 100 and 200. There are also the Tax Messengers, which hold between 100 and 500 points. We do not count them because we want it to be credible for this priority to be 100. To be 100, the players need 3000 gold. With the addition of the potential 1000 that Tax Messengers can provide, this became possible, even though difficult.

- *Time Since Arrested t2* - We could leave it at points, but for the sake of precision, we decided to take this, and the next variable, also into account.

  The match has 420 seconds, and applying the following equation, we will get:

$$t2 = \frac{player_howLongSinceArrested * 100}{gamelength/2} \tag{4.7}$$

  In our case, half of the game time is 210 seconds, so we feel that not being arrested for this long is terrific enough to warrant maximum priority. During testing, we did see players capable of being arrested only during the raid, as rare as it was, so this seemed to work.

- *Houses Robbed Without Being Caught t3* - The third and final part of the overall priority value is how many houses the player has robbed, in succession, without being caught. We decided to use one third of the number of houses as the point at which this part reaches maximum priority.

$$t3 = \frac{housesRobbed * 100}{maxNhouses/3} \tag{4.8}$$

  In our case, this means 10 houses in a row, without being caught, will put this part at one hundred percent.

Finally, we calculate the final local priority value, from all the parts, by giving each part a contribution to the overal value, as so:

$$priority = \frac{t1 * 80}{100} + \frac{t2 * 10}{100} + \frac{t3 * 10}{100} \tag{4.9}$$

Which means the points are worth ninety percent of the priority value, and the other two are worth five percent each. As much as we would like to further consider the other two values, how many points a player has really speaks decisively towards whether or not he is winning.

As mentioned before, as an added note, this is the metric the Shot Manager uses to consider which player is winning. So the heuristics "interest" and "survivor".

Ability Cooldown

This metric monitors the cooldown of an ability. It takes into consideration if there is more than one, and its priority grows as the cooldown counts down. Its heuristic value is fixed at "unnexpected".

Let x be the numerical id of the ability, tx is the cooldown value, rx currently cooling down value, and na the number of abilities, the priority value is calculated as so:

$$priority = \sum_{x=0}^{n} a(\frac{\frac{rx*100}{tx} * \frac{100}{na}}{100}) \tag{4.10}$$

Every ability is the same, and the heuristic is "unnexpected" because other players and characters do not know the ability is about to become active. The player himself does not know, since he does not have a numeric countdown like the HUD of the spectator has.

Ability Used

This metric monitors if an ability has been used. No ability can be used simultaneously, so the way this metric is implemented is that when an ability is used, it assumes maximum priority, retaining it until one second has passed.

The only thing it does, other than that, is check whether or not the ability is "PVP", which is to say, if it is an ability that harms an opponent, or if not, an ability that assists the player in some tactical way.

If it is "PVP", it assumes the heuristic value "conflict", otherwise it will assume the heuristic value "tactical".

Proximity to Objectives

This metric monitors how close the player is to the "objectives". In our case, we have treasures and raid exits as objectives.

We also take into account how much gold there is in the chest, if it is a chest, and allow raid exits to be worth more than the chests. How we do this is that each time we account for a chest in our formula, we count it as a part of the overall. So if there are 10 chests in the proximity, which never happens because there is only one treasure per house and our maximum interesting distance is approximately that, but if there were, how close we are to that chest would account only for ten percent of the overall priority. But a raid exit would have full value.

So there are two parts to this priority value; again, let us call them t1 and t2: t1 is the gold the chest(s) may have, t2 is how close we are to an objective. Plus, maxD is the distance at which we find it relevant to account for the objective, minD is the distance at which we reach maximum priority (when the player is close enough to interact), actD is the actual distance to the objective, tg is how much gold there is in the treasure and maxG is the maximum of gold a treasure chest can have

For each treasure found, we perform the following equations:

$$t1 \quad += \quad \frac{tg * 100}{maxG} \tag{4.11}$$

$$t2 \quad += \quad \frac{(actD - minD) * -100}{maxD} + 100 \tag{4.12}$$

And then we add the account of the raid exit by repeating the t2 equation above for it. There is always only one raid exit.

Now, with nT being the number of treasures that were close to the player than maxD, we calculate the final local priority value:

$$priority = \frac{t1 * 40}{100 * nT} + \frac{t2 * 60}{100 * nT} \tag{4.13}$$

The raid exit is not attenuated like the treasures. The actual proximity is worth sixty percent while the gold the treasures have is worth forty percent. This, as the in the other cases, seemed to work well during the initial tests. And it stands to reason since proximity to objectives is not as important as the name suggests, especially when the objectives have no interest (no points).

Finally, this metric has the fixed heuristic value “tactical”.

Proximity to Foes

This metric functions in a very similar manner as above. The difference is instead of gold, we look into the type of the foes, giving them fixed values.

We experimented and the following values seemed to work: 100 for other players, 80 for Tax Messengers, 60 for Pro Militia, 45 for Rat Militia and 30 for Standard Militia. There is no way this component will ever be 0, if the second is not 0.

The second component is the proximity. The math is exactly the same, even the fact the first component, enemy types, is worth forty percent while the actual proximity is worth sixty, but this time around, we add a modifier. This time, the more characters within proximity of the player, the more important it is in a more critical manner than with treasure chests. So if nF is the number of foes within maxD, the final local priority is calculated as so:

$$priority = \frac{t1 * 40}{100 * nF} + \frac{t2 * 60}{100 * nF} + nF * 10 \tag{4.14}$$

Lastly, the heuristic value is fixed on “conflict”.

Interaction

This metric monitors the interactions of the player with doors and treasures, and is composed by three parts, let us called them t1, t2 and t3.

Now t1 is the time we have been interacting, t2 is whether we are opening/unlocking or closing/locking and t3 is whether we are interacting with a door or a treasure.

Having tI as how long we have been interacting and iT as the total interaction time, t1 receives its value from the following formula:

$$t1+ = \frac{tI * 100}{iT} \tag{4.15}$$

t2 will be 50 if the player is closing and 100 if the player is opening. t3 will be 30 if the player is opening a door, 60 if the player is interacting with a treasure chest with money or closing a door, and 100 if the player is opening an empty treasure. This decision surged from discussion with the other experts on the game, the developers. Locking a treasure chest after it has been robbed is a good trap to lay, it makes the opposing player waste a considerable amount of time, so AutoBroad should capture the moment when said player realizes he has been fooled. Hence the 100 percent for t3 if the player is opening an empty treasure. Part of this trap is closing the door to the house, which is why we put that event as worthy as robbing a treasure, which is remarkably a commonplace event in a match.

The final priority value is calculated as so:

$$priority = \frac{t2 * 15}{100} + \frac{t3 * 35}{100} + \frac{t1 * 30}{100} \tag{4.16}$$

The interaction time is worth thirty percent, whether we are opening or closing is worth fifteen percent and what we are interacting with is worth thirty five percent, mainly due to the particularities we described, that we really wanted to have some weight.

Lastly, the heuristic value for this metric is always “tactical”, except for when the player is opening an empty treasure, in which instance it is switched to “unnexpected”.

Noticed By Foe

This metric monitors the lack of stealth from the player. It helps the metric “Proximity to Foes”, and is the most dynamic metric of them all.

It has two parts to it, again let us call them t1 and t2: t1 is, at the same time, what kind of character noticed the player as well as if it is pursuing him. And t2 is how long has it been since the player was first noticed by an NPC, assuming 0 everytime no NPC is aware of the player.

It functions in the same way as the two proximity metrics, in the sense we add to t1 for every character that is noticing the player, a value from 0 to 100, and then collapse the resulting sum into a value between 0 and 100.

As far as t1 is concerned; if the player is being noticed by a Militia Pro, t1 will sum 100, if he is being noticed by some other militia class (Rat or Standard), t1 will add 60. If a Tax Messenger has noticed the player, t1 will sum 80. It may seem odd the

priority is the same where the player is being pursued, or just noticed, but that is because we actually change the heuristic value to differentiate between those states. When the player is only noticed but not pursued (heard), the heuristic is "unnexpected". When the player is being pursued (seen) for more than one second, the heuristic assumes the value "conflict".

Finally, t2 sums once and singularly. Unlike proximity, where we consider the proximity of each character to the player separately, here, the player is either noticed or not, independently of how many characters are noticing him. t2 multiplies by a fixed value that we tested profusely.

While testing the game, we put the average chase time at about 3 seconds. One of the largest we saw were 10 seconds. The timer resets the moment the player either distances himself from the characters by 1.2 times his hearing radius, or hides so the character in question no longer hears or see him. The number we chose was 8. Through that, we allow for some leeway, as the player would have to be 2.5 seconds over the longest chase we observed (10 seconds) to reach value 100. Because of this, we were confident this equation with the value 8 would work well, and it has proven so during our tests.

Having nN as the number of characters currently noticing the player, we can calculate the final priority value.

$$priority = \frac{t1 * 40}{100 * nN} + \frac{t2 * 60}{100} + nN * 10. \tag{4.17}$$

Which characters are noticing the player are worth forty percent, how long the player has been noticed is worth sixty percent and we add ten for every character that is noticing the player, just like with the metric Proximity to Foes.

Status

This metric monitors if our character is afflicted (stunned or disoriented), or arrested, and how long until he gets out of it.

Initially, we mapped it straightforwardly: the heuristic value was fixed on "tactical", and we made the priority value start at 100, then descending to 0 as the time to recover from the affliction, or jail, elapsed.

The problem was that, more often than not, the system watched most of the jail time of the player, which is not ideal at all. On the other end, it left a stunned or disoriented player just as he was about to recover, missing on some pretty narrow escapes from militia NPCs.

Currently, the moment the player is stunned, or disoriented, his priority becomes 50 and the heuristic value switches to value "conflict", because the player was just stunned, or disoriented, by an opposing player. Then the priority will grow as the time to recover elapses. When the priority value surpasses 75, the heuristic value is switched to "unnexpected", for the same reason the metric Ability Cooldown has the heuristic value "unnexpected".

If the player is arrested, the priority assumes 100 again, but this time, it quickly descends, at double the pace, so it becomes 0 when half the time to escape elapses. Furthermore, it switches the heuristic value to "other", when the priority value goes below 75. This translates the notion that the system only wants to show a player was arrested, and then quickly leave.

These attributions guarantee the behaviour we want from this metric, and greatly increased the perception of the Shot Manager about what is interesting.

4.3.2 Game-State Interpreter

The Game-State Interpreter manages all the metrics, making sure they are all up to date every time the Shot Manager has to make a decision.

To accomplish this in the best way possible, we make math based off configurable variables to ensure that we spread out metric updating as much as possible, across ticks, but in a way they are all updated within the timed decision cycle of the Shot Manager. To be light on the system like this, means avoiding clustering processing on the same tick as much as possible.

To accomplish this, we have the following variables:

interpretingInterval: An interval of time we are sure to be either equal or bigger than any tick, but not big enough to hinder the main objective (spread the metrics around). We placed it at 0.1 seconds, since the longest tick we recorded was 0.074.

ShotManagerCycle: How often does the Shot Manager update? In our case, this was 0.5 seconds, a value decided upon amongst the developers, experts on the game. Other games might have other values, it is up to the developers and/or experts to know how often does the Shot Manager need to update to make sure he does not miss anything.

And these are the only values that need to be fixed, that are open to configuration. The rest follows from mathematical manipulations.

We divide the number of metrics between sub-sets we named "branches". Each branch is processed within "interpretingInterval", processing one metric per tick until "interpretingInterval" is reached, at which time we process what is left of the branch. It will be rare for the metrics to equally populate each branch so we have an extra one with which we fill the extra metrics. The idea is every time all the metrics in a branch have been processed, but "interpretingInterval" has not yet been fully depleted, we take metrics from that extra branch, taking advantage of the "free" ticks to process them.

Due to the fact there is no such thing as two-dimensional arrays in unreal script, we had to get creative with index math, but we do not feel it is worth getting into an in-depth explanation about it. What is important is the Game-State Interpreter spreads the metrics as much as possible in the way we have described: does its best to process as few metrics as it can, every tick, while making sure they are all updated within the ShotManagerCycle.

For example, in our case, we had 16 metrics and 5 branches; additionally, to remind, "interpretingInterval" was 0.1 and "ShotManagerCycle" was 0.5. So that is 3 metrics per branch plus an extra 1 ($16/5 = 3.2$). So every 0.1 seconds we have 3 metrics updated. On one of the five branches, which is to say within each "interpretingInterval", we will

have 4 metrics processed instead of 3, the 3 regular ones and the 4th extra. The fourth is processed either during a moment where ticks are faster, or at the end, being the last metric processed.

Though the real core of game interpretation is actually built into the metrics, the manager, as always, is the one to get the name and responsibility for the job. Its job being interpreting the state of the game match.

4.3.3 Shot Manager

The Shot Manager is the broadcasting director. Charged with over-viewing the broadcasting itself, it handles the Shots, picking out the most important ones to show, thereby commanding the camera system. In our case, there are only two shots.

Aware of the functioning of the whole system, it will apply the last modifiers to the priority values of the metrics, seeing them as bundles that contribute to the overall priority of a point of interest. As we have explained, there are two points of interest, focused on each player, and represented by the Shots. Each Shot has direct access to all the metrics that are related to the player it is watching. When the time comes for the Shot Manager to make a decision, every half a second, it will look into its shots, analyzing and drawing from each one, through calculations, a priority value. Then, knowing full well how important each shot is, the Shot Manager can make its deliberations about switching shots, while taking into account other factors.

So fist we will cover how to calculate the priority of a bundle of metrics related to a point of interest, or in our case, the priority of a Shot.

Shot Priority

The Shot Priority comes from the direct sum of all priority values from all the metrics, while applying the heuristics modifier to each one, while also transforming each value into a contribution to the Shot priority value, which also has to be between 0 and 100.

Let us discuss the second transformation first. The idea is each metric priority would account for 100/nMetrics. That would have been simple, but sadly insufficient. The overall priority value should not suffer the weight of the completely inactive metrics (metrics with priority 0), but compensation math should neither be fixed, nor random, it should match the kind of values the active metrics have. Let us look at some examples, so to better understand the problem:

Each shot has 8 metrics. Imagine Shot1 has 3 active metrics, all with priority value 100, while Shot2 has 6, all with priority values 50. With each metric being worth 12.5 of the total value, Shot1 has an overall priority value of 37.5 while Shot2 also has a priority value of 37.5. This is clearly not right. The first solution would be to use the number of active metrics as the total, instead of using 8. This makes it bearable, but not ideal. In this case, Shot1 would have a priority value of 100, which is dead wrong since there are more metrics that could be active, but are not, exactly because the shot is not that important.

We wanted these values under control. On one hand, we have Shots with priorities rarely leaving the 20s, because they are weighed down by the inactive metrics. On the other, we have Shots achieving high priority values like 70 and 80 too easily. We wanted the weighing down to happen, since it makes sense for the overall priority level to be affected by the lack of active metrics (theoretically, it is possible for every metric to be active. Practically speaking, most metrics seem to be active only at rare times, which is what makes those times interesting and worthy of a high priority value).

We performed the math for a lot of examples, and reached a solution which we believe has proven to be sound and fair. We decided Shots would have its high-value metrics count more than they should, for example more than 12.4, but still hold them in that limited domain, calculating low-value metrics within it.

The way we accomplished this was by calculating the average priority of the metrics. Before performing the final sum, we make one pass through the metrics, calculating the average priority value, but *ignoring inactive metrics/zero values*. Then, when we are applying the final sum, if the priority value of the metric is above this average, we apply a little modifier to it based on their difference.

At the same time, we apply the heuristic value modifier. With nMetrics being the number of metrics a Shot holds, mPriority being the local priority value of the metric in question, valueOf(nHeuristic) being the numeric value of the heuristic property of the metric and averagePriority being the average priority of all the metrics related to the shot that have values over 0, we would have the following:

For every local priority value that is above averagePriority, we add as so:

$$ShotPriority+ = priorityToAdd * \frac{100.0/nMetrics}{valueOf(nHeuristic)} * \frac{mPriority}{averagePriority} \quad (4.18)$$

And for every local priority value that is below averagePriority, we add as follows:

$$ShotPriority+ = priorityToAdd * \frac{100.0/nMetrics}{valueOf(nHeuristic)} \quad (4.19)$$

valueOf(nHeuristic) is calculated to four different values:

- “Conflict” - 100
- “Unnexpected” - 150
- “Tactical” - 200
- “Other” - 250

Once that is done, we still apply the very first idea, which is to translate the value into the main domain. With nActiveMetrics being the number of metrics with non zero priority values, we calculate:

$$ShotPriority = \frac{ShotPriority * nMetrics}{nActiveMetrics} \quad (4.20)$$

Finally, we apply the time modifier: if the Shot is not active (being shown), then we directly add up the amount of time that has passed since it was deactivated. Before doing so, we make sure that that value is lesser or equal than fifty percent of the ShotPriority. So basically, we add to the ShotPriority up to a maximum of fifty percent its value.

We decided on this solution due to the benefits of inverse scaling; the largest the ShotPriority, the more it can be increased and the longer it takes to reach the maximum potential of that increase. To remind, the time heuristic was defined by the tendency broadcasters have to switch shots, and show different things, to retain the interest of the viewer, which rapidly declines when looking at the same thing. The more important the Shot, the more the time heuristic will be a factor and the longer it will take to exaggerate. This seems ideal to us and it has shown to work sufficiently well.

Finally, there is the "survivor heuristic" to take into account. The Shot Manager accesses the metric "Competitive Stats" directly, to see which player is losing, and gives the appropriate shot a twenty percent boost to its priority value. It was thirty at first, but that proved to be too high.

Decision Process

Now that we know how the Shot Manager calculates the importance of a shot, we will explain how it actually decides to perform switches.

The way it works is that at every decision cycle, half a second in our case, the Shot Manager browses the inactive shots, updating their priority values, and theirs only. It then picks the one with the highest priority to makes its interest check.

It has four levels of interest checks, all controlled by a timer which tells the Shot Manager how long since it activated the currently active shot that has been active the longest (how long since he switched the shot that has been shown the longest, into being shown).

- *Critical Level Check*: Without time restrictions, which is to say, every half a second, it checks if Shot Priority is equal or greater than 92. If so, it performs a critical switch, which is a switch with the least important active Shot.

- *High Level Check*: If two seconds have gone by since it activated the oldest activated shot, and Shot Priority is equal or greater than 70, then we switch it with the least important active shot that also has been active for more than a second.

- *Mid-Level Check*: If ten seconds have gone by since it activated the oldest activated shot, and Shot Priority is equal or greater than 40, then it switches the Shot with the least important active shot that has also been active for more than a second, and only if that active shot has a lower priority value than the inactive one we are looking to activate.

- *Low-Level Check*: If twenty seconds have gone by since it activated the oldest activated shot, and Shot Priority is equal or greater than 20, then it switches the Shot with the least important active shot that has also been active for more than

a second, and only if that active shot has a lower priority value than the inactive one we are looking to activate.

- *Crisis-Level Check*: If one minute has gone by since it activated the oldest activated shot, regardless of Shot Priority, it will switch the Shot with the least important active shot that has also been active for more than a second, and only if that active shot has a lower priority value than the inactive one we are looking to activate.

We compare inactive shot priority with active shot priority when we are looking to activate it. We look only at the time and inactive shot priority, before deciding we want to activate our shot. Only then do we run through the list of active shots, calculating their current total priority so we can compare and finally decide whether or not to make the switch.

These numbers and decisions have good and bad points, and most if not all the bad points would heavily be dissuaded through the use of viewport management.

The worst point is what happens when two shots have critical priorities, difficult as that may be; what happens is that the system will be switching between them every second, and in all honesty, that is exactly what we want in this replay-lacking application of AutoBroad. This is how the professional broadcasting teams would behave, should they be faced with two very important points of interest, happening concurrently. We observed these in some matches during preliminary research, and this is what the behaviour we observed.

The other dangerous point is high-level check versus priority-level check. It would occur, and it did to us, that the high-level check could very well interrupt a critical point of interest at a critically important time. While that may be possible, more often than not, a high-level priority value will increase, or lower, its rank in the very near future, more commonly rising to critical-level. It is actually a maneuver to predict a critical-level situation. If the system had it wrong, and the high-level decreases to mid-level, or even stays high-level, no major harm was done since the system will only look away from the original critical-level situation for one second, since it will switch right back after that time. But still, one second can be very important, and the timing interruption, difficult to predict as it is, may be absolutely wrong. And we agree: such is a fault of any broadcasting system lacking replay functionality or any other alternative to more directly deal with concurrent events.

Finally, in terms of our implementation, the Shot Manager also serves as a distributor of information. It uses a camera proxy (not a Shot, an actual Camera) on its side of the network, server side, to make the calculations. It is to it that the Shot Manager passes the active shots, and once that proxy has calculated the location and rotation and other camera configurations, the Shot Manager distributes that information to every camera belonging to every spectator, client side, that is connected to the system. It is also responsible for retrieving game information to provide the client-side HUD of every spectator, since UDK makes it impossible for every other class to do so.

Quickly getting into low-level information, UDK only replicates information through

"player" classes. Shot Manager was, thus, made to be a player class, and every replication has to go through it. Any other solution that we looked into would have the system suffering latency issues over communicating a few strings and booleans across the network.

Concluding

Lastly, there is only the Camera System to look at. There is actually nothing to explain, on this module, that has not been explained already. We have only one camera per spectator, indirectly tuned in to a "stream" of information provided by the server-side camera owned by the Shot Manager. There is no viewport animation since UDK does not give that kind of control and there is also no replay buffer, or implemented behaviour, in that sense. What the camera does is switch between player locations, applying the extra transformations to offer the isometric view that players share. It does that based on its active shots, which are provided by the Shot Manager. In our case, only one is ever provided.

We really decided to focus all of our effort into accomplishing game-state interpreting well. As it stands, any player based TPS game could use the Game-State Interpreter and Shot Manager described here, managing and working with metrics defined specifically for said game, and AutoBroad would be broadcasting that game just as well as it does Shadow Conclave.

In the next chapter, we will see just how well AutoBroad does just that. We will discuss what tests we have made, what results they drew and what conclusions we were able to gather towards the performance of AutoBroad.

Chapter 5

Evaluation

In order to establish how successful our system really is, we took it upon ourselves to test it with users. In this part of the document, we will describe how we did so, and what results we were able to draw.

5.1 Testing

With the system good and ready to be used, we quickly patched it with a "manual behaviour". This behaviour, or version, allows the spectator to control which player he is watching. It is critical to be clear about the fact that the users did not moved or controlled the camera, but were simply doing the job of AutoBroad, dictating who the camera showed. They controled it with the spacebar, switching between players at a press. Since we sought to compare performances, it would be unfair for the manual mode to be harder to control.

And that was the goal: to compare how well a normal manual control makes its decisions when opposed to AutoBroad. We separated the overall success of the system into two parts: Functional and Quality. We try to see which system shows a better functional result (makes the right decisions and shows the spectator what he should be seeing in order to understand the match), and we try to see which system delivers the best quality experience (which do they like best? Automatic or Manual?).

To achieve this with a good amount of certainty, our testing was divided into two phases.

5.1.1 Phase 1 of Testing - Diversity And Unfamiliarity

In this phase of testing, each tester was to watch two matches with each broadcasting version (Automatic and Manual). After each match, they had to answer a series of questions to determine the Functional and Quality success of the system they used. The key idea in this testing was to simulate viewers which were unfamiliar with the game, or even gaming in general. We only informed each tester of the following information, word for word: "It is an automatic broadcasting system. You only need to pay attention to

the game it is casting. The video game is about thieves who are competing to see who steals more money, while trying to avoid being arrested."

We recorded each match, not what was shown by whatever system the tester was using, but both views of the players. From both videos, we retrieved the real answers to questions that we ask, so we could later grade the answers filled by the respective tester. We also highlight that each match was different, played by experts on the game (the game developers).

5.1.2 Phase 2 of Testing - Common and Familiar

In this phase of testing, we had recordings of the broadcasting of two matches, one by AutoBroad, another by a tester of the first phase. We divided all the testers at the middle, to the best of our ability, and they watched the recording. This time, though, before they watched it, they were informed about the game using a quick lesson picture, which can be found in the appendix 6.1: it is a short lesson, teaching just enough to simulate a familiarity with Shadow Conclave, not an expertize.

After watching the recording, they would fill out the same questions except for one, as we will see below.

We believe through this manner, we would really retrieve the truth of how our automatic system held up against a manual one, which to remind, is the "state of the art" method to broadcast video games.

In closing, we cover individual influence by testing many people, in general. We test match influence by testing one match per tester in the first phase, and testing one match per group of testers in the second phase, and we cover familiarity by giving it to a group of testers, in the second phase, but not to the other, the first phase.

Finally, we will present the questionnaire that our testers were asked to answer.

5.1.3 Questionnaire

We divided our questionnaire into two types of questions, though we did not order it by type of question so not to complicate the line of questioning itself.

Match-dependant questions are questions which answers entirely depend on the match, and so are dependant on the perception of the match, which is majorly dependant on what the broadcasting system shows the tester. We use this to evaluate the system functionally.

Match-independent questions are questions which answers entirely depends on nothing but factors of the system itself which are not, in any way, relying on what is happening in the match. What it shows, what it sounds, how it looks, etc. We use this to evaluate the system in terms of quality.

Each question has a multiple choice, and we attribute each choice a numeric value. This is done so that match-independent answers can be added, reflecting the success of the system in terms of quality by how high the resulting value is. Match-dependant

answers have values attributed to them which will reflect how right they are by how low the value the comparison method deals; and the method is that we calculate the direct difference between the given answer and the real answer, and the wronger the given answer is, the greater the difference will be, thus reflecting the success of the system in terms of functionality by how close to zero the total value is.

How we arrange for the real answers is by having an expert carefuly examine the recording of each match from each player's perspective, filling out the questionnaire.

The difference between phase 1 and phase 2 is but two questions. In phase 1, we asked, upon the second test for each tester, which of the systems they preferred (Automatic or Manual) and why. On the second phase, we replaced this question by how many times they watched the recording of the match.

We will now show the list of questions regarding phase 1, along with the numeric values they are attributed to, and what type of question they are(match dependent or independent.):

- *1 - How much would you say you enjoyed the match you saw?*

  Match-independent answer going from 1 to 5.

- *2 - In your opinion, how visible was the information displayed on the screen, about the players?*

  Match-independent answer going from 1 to 5.

- *3 - In your opinion, how useful was the aforementioned information, towards the general understanding of the match?*

  Match-independent answer going from 1 to 5.

- *4 - Which player would you say showed more skill?*

  Match dependant, answers were:

  - Player 1: Hunter - attributed 1

  - Player 2: Circus Freak - attributed 2

- *5 - Which of the players actually won the match(ended up with more points)?*

  Match dependant, answers were:

  - Player 1: Hunter - attributed 1

  - Player 2: Circus Freak - attributed 2

- *6 - Select below the set of words you feel better classify the style of play of the Player Hunter:*

  Match dependant, answers were:

  - Patient, Careful, Prudent - attributed 1

  - Impatient, Careless, Reckless - attributed 2

- *7 - Select below the set of words you feel better classify the style of play of the Player Circus Freak:*

  Match dependant, answers were:

  - Patient, Careful, Prudent - attributed 1

  - Impatient, Careless, Reckless - attributed 2

- *8 - When arrested, the players suffered consequences. Select below the option you feel best defines the consequences they suffered upon being arrested.*

  Match dependant, answers were:

  - They only lose time. - attributed 1

  - They lose a fixed quantity of money. - attributed 2

  - They lose a quantity of money relative to what they have. - attributed 3

- *9 - Which of the players do you feel robbed more treasures?*

  Match dependant, answers were:

  - Player 1: Hunter - attributed 1

  - More or less the same. - attributed 2

  - Player 2: Circus Freak - attributed 3

- *10 - Which of the players do you feel got arrested more?*

  Match dependant, answers were:

  - Player 1: Hunter - attributed 1

  - More or less the same. - attributed 2

  - Player 2: Circus Freak - attributed 3

- *11 - Did you notice Player Hunter employing any skills?*

  Match independent, answers were:

  - No - attributed 1

  - Yes - attributed 2

- *12 - Which of the abilities did you see Player Hunter use?*

  - Value Sensing(Stays still, talking) - attributed 1

  - Stun(Immobilizes other characters) - attributed 2

  - Both - attributed 3

- *13 - How many times would you say Player Hunter used the ability "Stun"?*

  - Less than three 3 - attributed 1

  - Between 3 and 7 - attributed 2

- Between 8 and 12 - attributed 3

- Between 13 and 17 - attributed 4

- More than 18 - attributed 5

- *14 - Did you notice Player Circus Freak employing any skills?*

  Match independent, answers were:

  - No - attributed 1

  - Yes - attributed 2

- *15 -Which of the abilities did you see Player Circus Freak use?*

  - Break an Entering(Opens a door much faster.) - attributed 1

  - Fear(Frightens other characters, making them move around chaotically) - attributed 2

  - Both - attributed 3

- *16 - How many times would you say Player Circus Freak used the ability "Fear"?*

  - Less than three 3 - attributed 1

  - Between 3 and 7 - attributed 2

  - Between 8 and 12 - attributed 3

  - Between 13 and 17 - attributed 4

  - More than 18 - attributed 5

- *14 - Did you clearly hear and understand the announcer?*

  Match independent, answers were:

  - No - attributed 1

  - Yes - attributed 2

- *18 - How many "Tax Messengers" did you see during the match?*

  Match-independent answer going from 0 to 5.

- *19 - How many "Tax Messengers" would you say escaped, during the whole match?*

  Match-dependant answer going from 0 to 5.

- *20 - In front of each player's name, please put the number of "Tax Messengers" you feel they respectively caught/robbed, during the match?*

  This one is the only exception to the valuing rule. It is a match-dependant question, and the answers can be these:

  - Hunter - 0 through 2

  - Circus Freak - 0 through 2

It is a question which numerical values ranges between 00(0 on both), 01(0 on Hunter, 1 on Circus Freak),10(1 on Hunter, 0 on Circus Freak),11(1 on both),20(2 on Hunter, 0 on Circus Freak) or finally 02(0 on Hunter, 2 on Circus Freak). So they were then represented, in terms of decimal number, as 0.1,10,11,20 and 2. The difference between the real answer and this given one was not straight math but, on this case only, a multi-dimensional "IF" clause that gives it a value between 0 and 4, depending on how far apart the real answer is from the given one.

- *21 - Which of the players would you say was involved in more pursuits?*

  Match dependant, answers were:

  - Player 1: Hunter - attributed 1

  - More or less the same. - attributed 2

  - Player 2: Circus Freak - attributed 3

- *22 - Did you notice the ending phase of the match, entitled "Raid"?*

  Match independent, answers were:

  - No - attributed 1

  - Yes - attributed 2

- *23 - Select the player(s) that managed to escape the raid completely un-caught.*

  Match dependant, answers were:

  - Hunter - attributed 1

  - Both - attributed 2

  - Circus Freak - attributed 3

- *24 - Which of the versions did you prefer?*

  Match independent, answers were:

  - Manual - attributed 1

  - Automatic - attributed 2

- *25 - Please offer some reasons towards your reply. Feel free to occupy whatever space you require:*

Question 24 and 25 were only answered when the tester had used both, which is to say, when he was on his second test (Phase 1). In phase 2, the question that replaced those two, "How many times did you review the match recording to answer these questions?", had the same answer from all testers: less than 3.

Now that we have covered how we performed the testing sessions, we will cover the results and conclusions we managed to retrieve, in the next section.

5.2 Results

The first phase of testing received the support of 17 testers, generating 33 total tests, representing close to 30 matches. The second phase saw 26 testers. Four signed up to do it, but ended up not participating so we, unfortunately, received more questionnaires on the manual system than automatic.

We expected from the tests, in general, for the manual system to be more enjoyed than the automatic system, and the automatic system would work equally well, or better, than the manual system, in terms of relaying the match. Reasons for the first expectation were on account of being a simple one versus one match and so not so stressful to follow manually, and for that reason, total control and power of choice would be preferred. Reasons for the second were on account of, again, being a simple one versus one match and so not so difficult to make the right choice and follow both players closely. This is why we decided to test this very first implementation of AutoBroad with this game.

We will now cover our deductions and conclusions.

5.2.1 Phase 1 Results

Functional Analysis

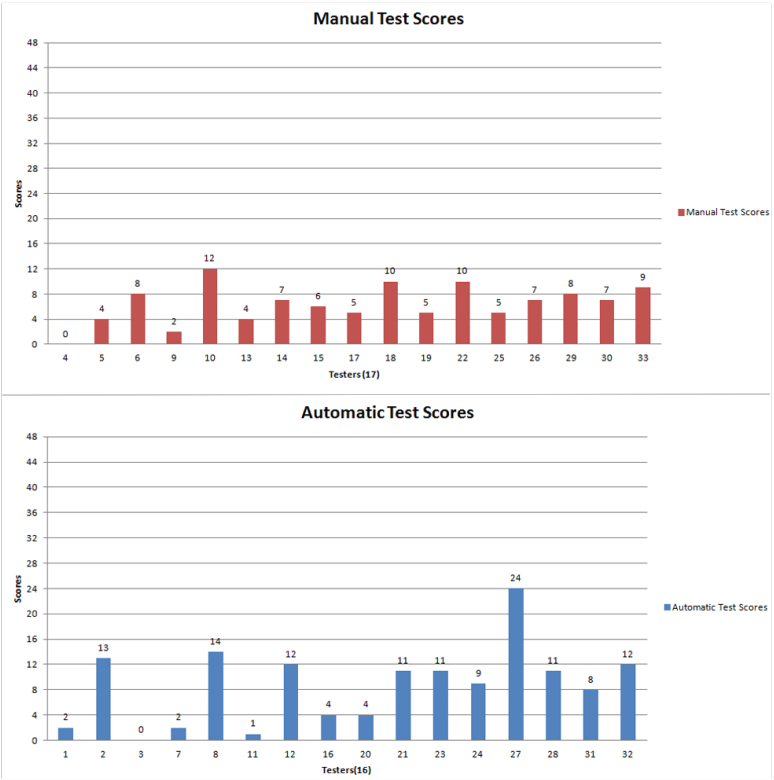


Figure 5.1: Test Scores, by Tester.

The test results can be seen in figure 5.1. The numbers across the horizontal axis are the IDs of the tests. Mostly, two tests represent a user.

In terms of functional results, 48 would be the grade they would have if they answered nothing. 33 would be the grade they would have if they answered everything, but as wrong as possible. 0 is the best grade possible. We will be comparing always to both numbers, since there was a total of 94 unanswered questions in all 33 questionnaires (out of 792).

Using the Manual mode, Testers scored an *average of 6.4*, with a *standard deviation of 3*, worst grade being 12, best being 0, with a *mode and mean of 7*.

Using the Automatic mode, testers scored an *average of 8.6* , with a *standard deviation of 6.2*, worst being 24, best being 0, with a *mode and mean of respectively 11 and 10*. The next worst grade, after 24, was 14. So considering that test an outlier, the values we then have are an *average of 7.6*, with a *standard deviation of 4.8*, worst being 14, best being 0, with a *mode and mean of respectively 11 and 9*.

The *average grade is between 86.6/80.6 (48/33) percent for Manual mode* and, ignoring the outlier, *an average grade between 84.1/77.0 (48/33) percent for Automatic mode*, respectively with a worst grade lying between 75/63.6 (48/33) percent and, ignoring the outlier, 70.8/57.6 (48/33) percent.

In total, they differentiate by 2.2, which translates to 2.5/3.64 (48/33) percent, which is approximately one question and a half. If we ignore the outlier, then the difference drops to 1.2, which translates to 2.5/3.64 (48/33) percent, which is approximately one question, if that. This means that, statistically, the worst that can happen is for the manual version to offer the tester the edge of getting right one extra question.

So our conclusion from these values are that results were sufficiently approximate, especially when taking into consideration that manual testing received three more "second-hand testing", which is to say, testers which tested Manual Version on their second try, and Automatic on their first try.

But this was a look at the results by tester. Looking at it by question, we would perhaps arrive at a different result, or draw new insight into the qualities, or flaws, of the system.

Figure 5.2 shows the average grade for each question, against the worst possible with no answer (sum of which would be 48) and the worst possible with an answer (sum of which would be 33). We can plainly see that not only do automatic and manual answers stick close together, but their values always stand below half of the worst. This means that in average, the questions are answered correctly.

The graph below, in figure 5.2, shows the non-absolute difference between manual and automatic average scores for each question. Negative values mean the Manual Version was more correct, positive values mean testers did better with the Automatic version. Only one question has a score on which they really differentiate. Question 23 involved answering how many players fled the raid. Unfortunately, for reasons related to the video

game and a certain bug or two the version we still used had, this is understandable. It was a very hard question to answer correctly, even if the system did its job perfectly.

In average, results are approximate and portray a good understanding of the game,

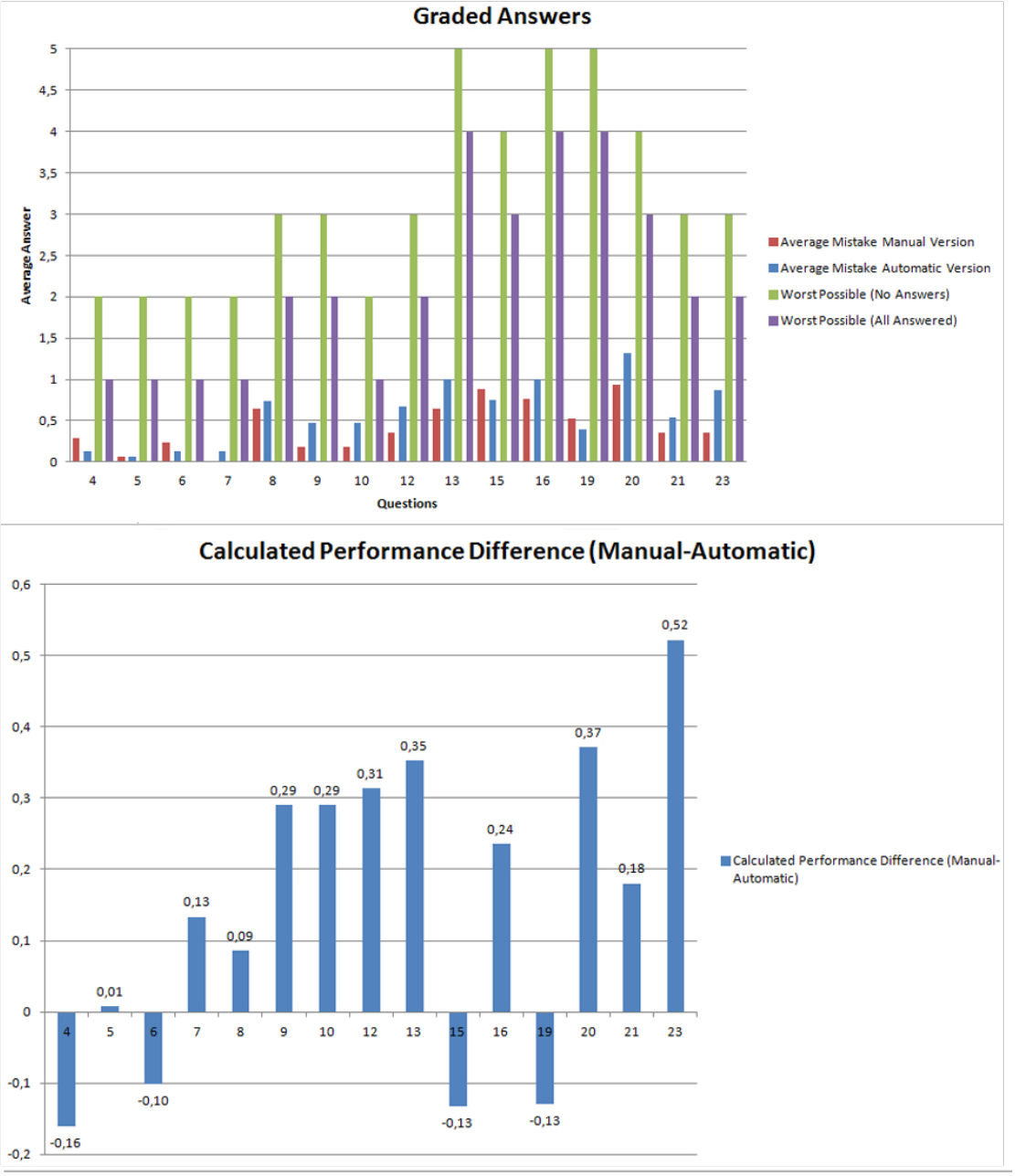


Figure 5.2: Test Scores, by Question.

at each question. The average of question 23, though, is the exception: with manual testers having one of 0.3 and automatic testers having one of 0.8, *it is clear, from this first phase of testing, that the system must consider raid exits even more important.*

Quality Analysis

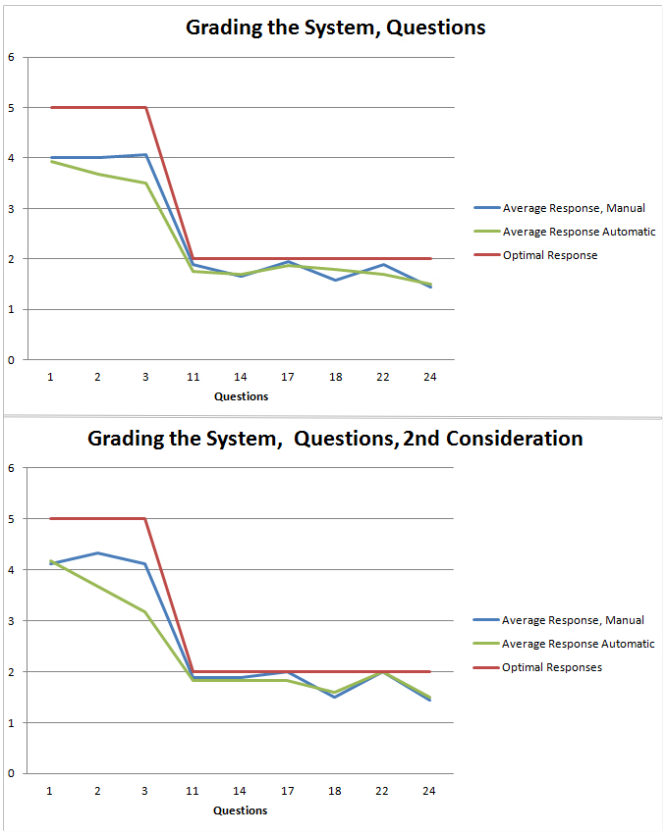


Figure 5.3: Test Scores, by Question.

Figure 5.3 shows the answers, by question, to the questions which rated the enjoyability of the system. The larger and closer to the optimal values, the better the grade. One can see both versions are very close to optimal values, and for that matter, very close to each other. The "2nd consideration" is basically an analysis only taking into consideration the answers that second-hand testers gave, which is to say, we take into consideration only the tests respective to testers who were functioning with the system for the second time, and were thus more familiar with both the system and the game. Improvement is noticeable, as we expected. Once the tester is familiar with the system, he or she will enjoy it more, which is always a good conclusion which will carry over to the results we have, by testers.

The most important fact is, though, that testers found the system enjoyable.

Figure 5.4 shows the test results by tester. The "2nd consideration" graphs mean the same as previously, which is why the horizontal axis is shorter, including less testers.

This time around, the grade had to be as large as possible, 29 being the best possible grade, and 0 the worst. We add each question score up to whatever total we get.

It is visible that the automatic version is once again plagued by an average ruining grade, 10. If not for this 10, the average of the automatic version grade would actually surpass that of the manual version, as we will see right now. For this reason, we dismissed it as an outlier, though since there is only one, we will, like before, offer the calculations including it. We are fans of presenting worst cases, so we can show not even those are bad.

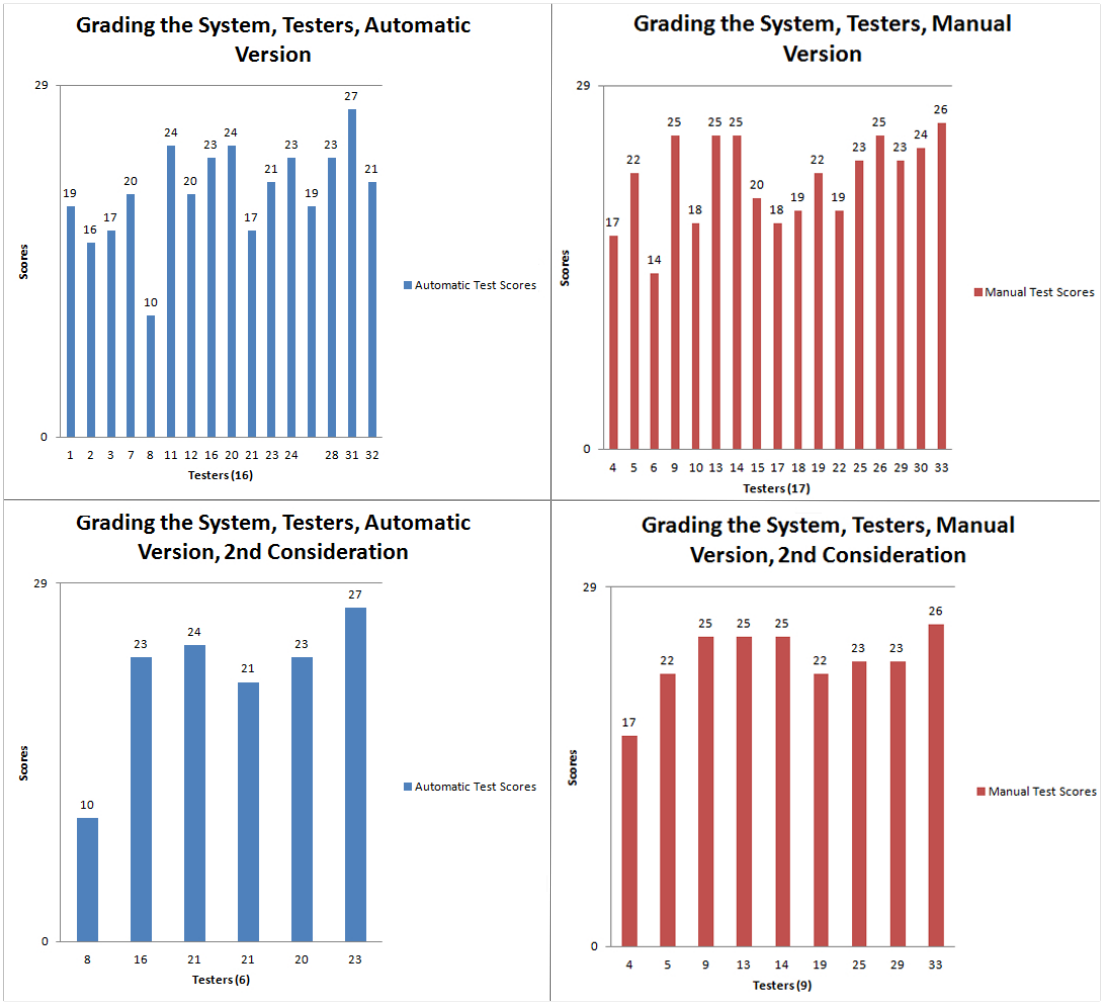


Figure 5.4: Test Scores, by Tester.

The Manual version, first consideration, has an *average score of 21.5*, with a *standard deviation of 3.5* and the worst score is 14. On the second consideration, it has an *average score of 23.1*, with a *standard deviation of 2.7* and the worst score is 17.

On the other hand, the Automatic version, counting the grade 10, has an *average score of 20,25 and 21.3, respective to consideration*, with a *standard deviation of 4.0 and 5.9, respective to consideration* and the worst score is 10 on both considerations. If we ignore the tester who gave it a 10, then the *average score is 20.9 and 23.6, respective to consideration*, with a *standard deviation of 3.1 and 2.2, respective to consideration* and the worst score is 16 on the first consideration and 21 on the second.

In the textual answer as to why they preferred one or another, in general, testers said that though the automatic version often switched players when they did not want too, the fact was they did not know enough for their preference to be justified. When using manual mode, they have the control, but the preoccupation that the other player might be experiencing something more interesting is heavy on their conscience. It is much more comfortable to have someone else hold the wheel. Those who preferred the manual version just preferred to have control.

All and all, the system seems to have provided a quality experience on both versions, and the Automatic Version did etter than we anticipated.

Still, if we received this kind of feedback for a one versus one match, one can only imagine how much better people would enjoy the automatic version on a free for all four player, or even eight player match. This if the automatic version performs as well, functionally, as it did here.

5.2.2 Phase 2 Results

In this phase of testing, we had two groups of testers. Each watched the recording of a match, unaware of whether it was from the automatic version or a manual one. Having the recording, we also evaluated what answers the recording showed, in order to fully evaluate the tests; this way we know how much the system, manual and automatic, influence the game perception of the user, which is to say, how well they function. We felt it was important since in this phase, a large number of testers watched the broadcasting of a single match, so it is important to see if the broadcasting "fed" them wrong answers.

Functional Analysis

Figure 5.5 shows the test results. The numbers across the horizontal axis are the IDs of the test. This time, each test represents one individual.

In terms of functional results, again, 48 would be the grade they would have if they answered nothing. 33 would be the grade they would have if they answered everything, but as wrong as possible. 0 is the best grade possible. We will be comparing always to both numbers, since there was a total of 46 unanswered questions in all 26 questionnaires (out of 598).

We include the score the tester would have if the answers sought after were those shown by the recording. "Shown" score is the score of what we are sure the system saw, while the normal score is of what really happened. The "shown" score was obtained by answering the questions with the aid of the recording, much like the tester did, but it was an expert which filled out the answers.

So using the Manual mode, evaluated against the real match, Testers scored an *average of 9.9*, with a *standard deviation of 3.0*, worst being 17, best being 5, with a *mode and mean of 8 and 9 respectively*. When evaluated against the shown match, testers scored an *average of 9.6*, with a *standard deviation of 2.7*, worst being 15, best being 7, with a *mode and mean of 10*. The difference is about 0.3, so the "Show" match would not be cheating the tester of even one full question, statistically speaking.

Using the Automatic mode, testers scored the same thing for shown and real (we will see why next), with an *average of 9.6*, a *standard deviation of 3.7*, worst being 16,

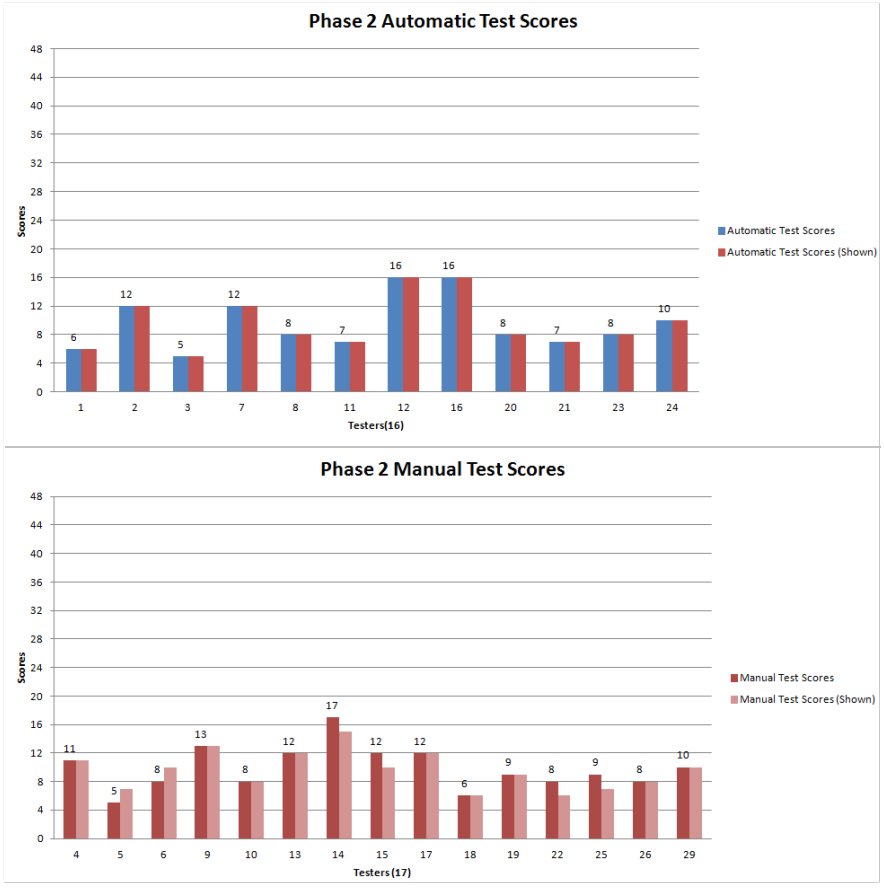


Figure 5.5: Test Scores, by Tester.

best being 5, with a *mode and mean of 8*. So, only looking against the real match, the *average grade is between 79.4/70.0 (48/33) percent for Manual mode* and *an average grade between 80.0/70.9 (48/33) percent for Automatic mode*, respectively with a worst grade lying between 64.6/48.5 (48/33) percent and 68.75/54.5 (48/33) percent.

In total, they differentiate by 0.28, which translates to 0.6/0.8 (48/33) percent. This means that, statistically, there is no advantage either way.

Our conclusion towards these values are that results were practically identical, even taking into consideration Manual testing already had the drawback of the "Shown" match being different than the "real match". However slightly, that tipped the advantage in favor of the automatic version into a tie, showing that AutoBroad functions better than an average user employing manual control, even in as simple a setting as one which offers the user, at any given time, a fifty percent probability of making the right choice.

But this was a look at the results by tester. Looking at it by question, we would perhaps arrive at a different result, or draw new insight into the qualities, or flaws, of the system.

Figure 5.6 shows, for both versions, the answers acquired from the actual match, the recorded match, and the average of the answers given.

The Shown match, of the manual version differs from the real match in questions 9, 19, 20 and 21. But how much?

Figure 5.7 shows an answer comparison by question. With it, we can use absolute difference to arrive at how different the Shown Manual match and the Real Manual match actually are.

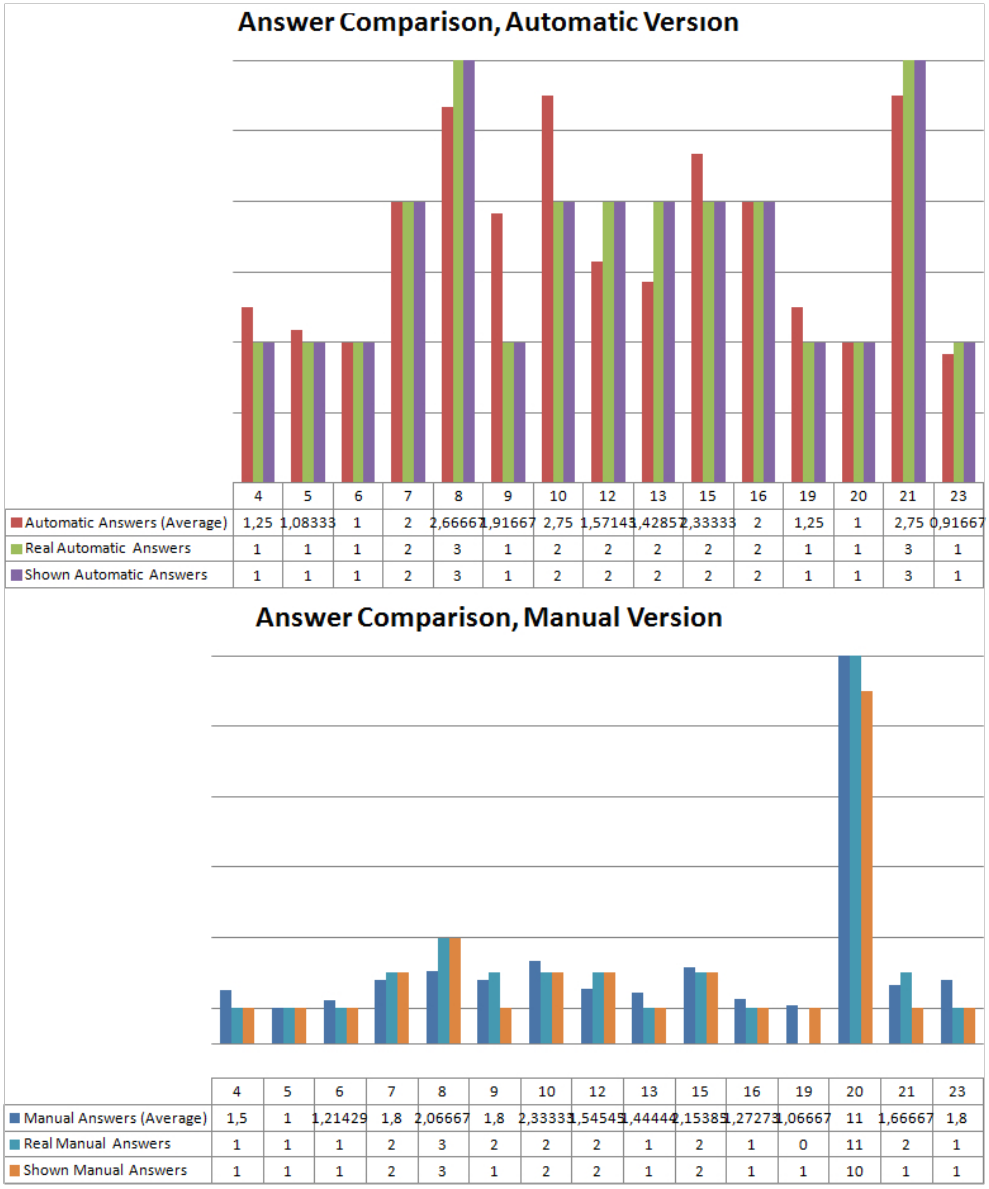
The absolute difference between the average of the responses, when put against the Shown and the Real match, is 0.6 for question 9, 1 for question 19, 0.2 for question 20 and 0.33 for question 21, accumulating at a total of 2.133. This means that the Manual version is actually cheating the tester of approximately one question and a half, perhaps two. But it was doing it "under the radar" because the testers happened to provide answers which were largely in favor of the Real Match, in question 20, and then answers which were variably in favor of the Shown match on the other questions. The non absolute difference of the questions(Real - Shown) draws the values -0.6 for question 9, 1 for question 19, -0.2 for question 20, and -0,33 for question 21, accumulating at a total of 0.13, which is a negligible value that explains the small difference in the test scores average.

Figure 5.8 shows the performance comparison.

The questions which have negative values differentiate in favor of the Manual Version while the positive values favor the Automatic Version, just like before.

The results are about the same as in the first phase, with the exception the Automatic Version simply did better this time around. At the exception of question 23, which we have already discussed, and 9 and 19, which were mainly caused by the fact the Shown Match did not represent the Real Match.

Question 9 is about who stole more treasures, and the reason it diverges so much is that the user whose Manual control was recorded saw a player steal more treasures when,



	4	5	6	7	8	9	10	12	13	15	16	19	20	21	23
Automatic Answers (Average)	1,25	1,08333	1	2	2,66667	1,91667	2,75	1,57143	1,42857	2,33333	2	1,25	1	2,75	0,91667
Real Automatic Answers	1	1	1	2	3	1	2	2	2	2	2	1	1	3	1
Shown Automatic Answers	1	1	1	2	3	1	2	2	2	2	2	1	1	3	1

	4	5	6	7	8	9	10	12	13	15	16	19	20	21	23
Manual Answers (Average)	1,5	1	1,21429	1,8	2,06667	1,8	2,33333	1,54545	1,44444	2,15385	1,27273	1,06667	11	1,66667	1,8
Real Manual Answers	1	1	1	2	3	2	2	2	1	2	1	0	11	2	1
Shown Manual Answers	1	1	1	2	3	1	2	2	1	2	1	1	10	1	1

Figure 5.6: Comparing the answers.

in actual fact, the two players stole about same (the recording shows an 8-4 relation when it was actually 14-13). The automatic version did not suffer this problem (it showed 11-7 when it was actually 18-11, so the relation was maintained).

Question 19 is about how many Tax Messengers escaped. They were both caught, so none escaped, but the manual version only showed 1. The Automatic version successfully showed both Tax Messengers being caught.

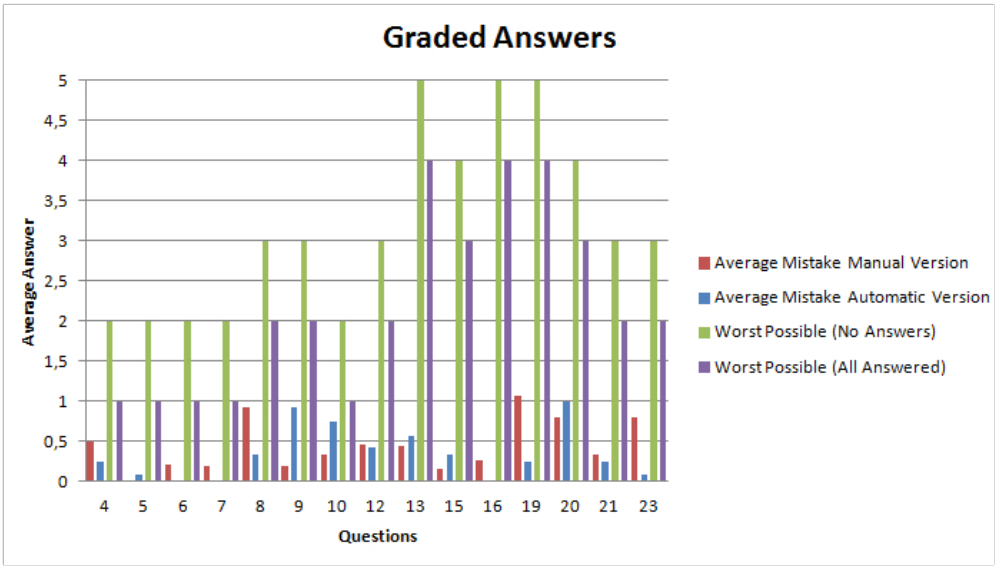


Figure 5.7: Comparing the answers, average of grades by Question.

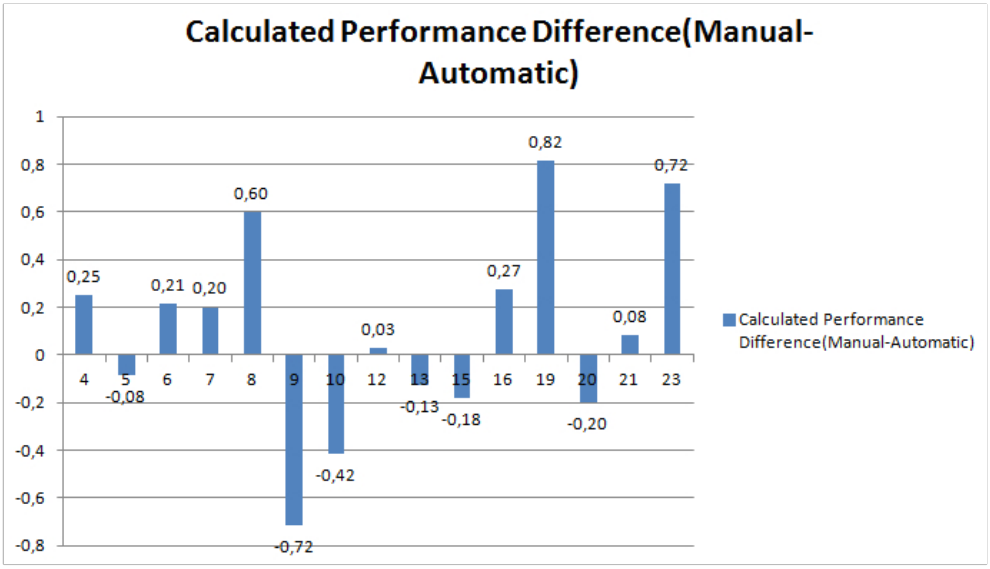


Figure 5.8: Comparing the overall performance, for each question.

Since this performance comparison is made against the respective Real Matches, these two differences proved proved to be acute.

Before we discuss general conclusions, let us discuss the final results of the quality tests during Phase 2.

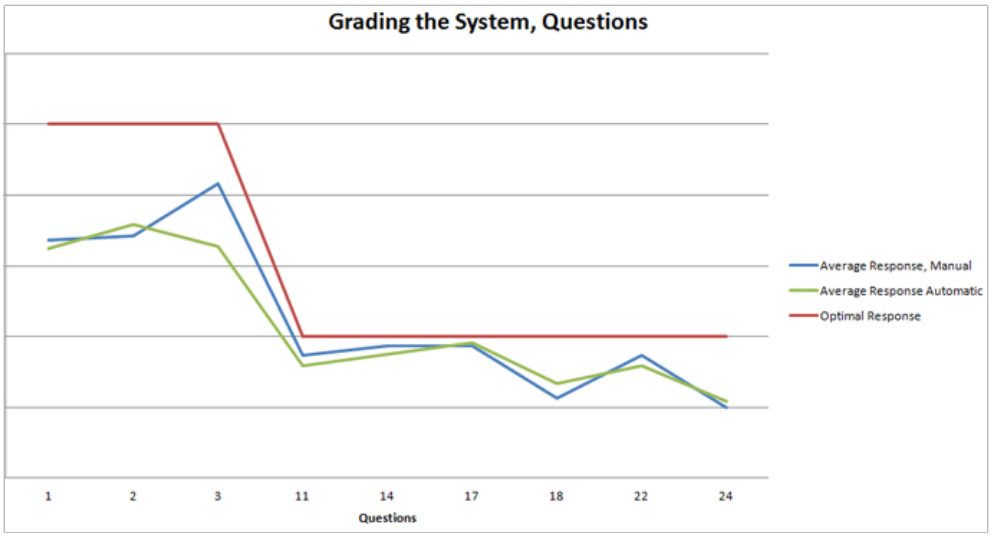


Figure 5.9: Test Scores, by Question.

Quality Analysis

Figure 5.9 shows the answers, by question, to the questions which rated the enjoyability of the system. The closer to the optimal value, the better. Both versions are very close to optimal values, and for that matter, very close to each other. They are, though, inferior to the values we retrieved during phase 1. We suppose the first impression really is not the best, as much as we try to make testers familiar with the game through a quick explanation and images, it is not the same as experiencing it, so what we believe happened is that instead of having 33 tests in which 15 are second-hand testers, experiencing the system for the second time, we had 27 tests in which about 20 were testers experiencing the system for the first time.

Also, second phase testers did not really have control, since they were just watching the match, as they would an automatic broadcast, which explains the natural tie. We will see next if that is maintained.

Nevertheless, results are positive since our HUD is a purely functional one, not the least bit worried about presentation.

Figure 5.10 shows the grading by Tester.

As with phase one, the quality grade had to be as large as possible, 29 being the best possible grade, 0 the worst. We add each question score up to whatever total we get.

This time, the manual version had the outlier: 11.

The Automatic version showed an *average score of 20.7*, with a *standard deviation of 4.0*, the worst score being 15.

On the other hand, the Manual version showed an *average score of 21*, with a *stan-*

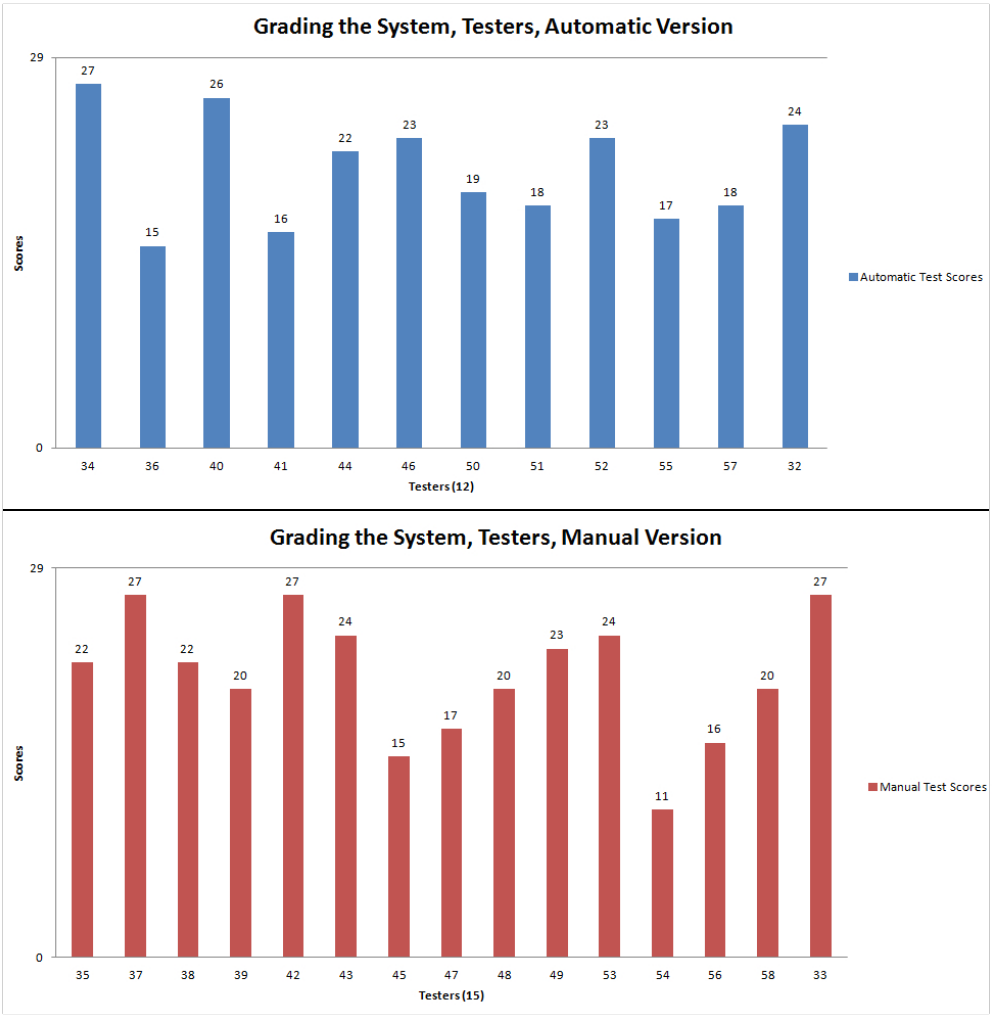


Figure 5.10: Test Scores, by Tester.

dard deviation of 4.7 with the worst score being 11 and the best 27. If we ignore the tester who gave it an 11, then the *average score is 21.7*, with a *standard deviation of 3.96* and the worst score would change to 11.

Conclusions remain about the same as previously seen: a clear tie.

5.2.3 Analysis of Results

- AutoBroad can functionally match the Average manual user.
- Second-experience testers have a better understanding, and likability, of the system than first-experience testers.

- AutoBroad will function equally well whether it is the first time it is being used, or the hundredth time. This is how and why it fared better than the Manual version, in the second phase of testing.
- AutoBroad is still not perfect. Not only did it not produce perfect scores, but scores below those shown by the Manual version, however small the difference was, *no more than a question.*
- AutoBroad should be tweaked towards raid exits, since that was the question where results were worse.
- AutoBroad exhibited no issues in weight, whether it be processing, latency, or memory.
- AutoBroad is sufficiently good quality experience.

Chapter 6

Conclusion

6.1 An Overview

During the course of our work, we sought to design an automatic real-time broadcasting system for video-games, and then to build one for a specific video-game. We established a number of features we feel a system like this should have, and then we focused on those features that have never been directly researched before, and thus done, as far as we know. We focused on making sure the system could automatically interpret the game-state of a match, with the goal of knowing what it should show the spectator so he or she do not miss any important event.

To accomplish this goal, we first sought out to study related work that could be useful in helping us devise a solution, we looked into works on the domains of strategy prediction, intelligent camera, event highlight in sports broadcasting and camera profiling. From those works, we chose to use the same approach as a work done on strategy prediction [MOR08].

The approach meant translating low-level information into high-level information, then interpreting that high-level information into a kind of ontology that represented the importance of events, or points of interest, from the point of view of a broadcasting system. To design that final interpretation, we committed to a meticulous observation of a number of broadcasted E-Sports matches, all from high-profile championships, involving high-profile players and broadcasters. We thus defined the heuristics which the system would use to tell how important a given point of interest was, from a broadcasting point of view.

With that done, we designed a basic model of a system that can be used and applied to any video-game. Then we applied that model, specifying it and then implementing the system on a working third person strategy video-game, Shadow Conclave.

It went through preliminary testing phases to help optimize it, involving the experts behind its development.

Following that, we evaluated the system by developing a version of a broadcasting system which was controlled manually, and thus we tested both versions with several testers, gamers and non gamers alike.

From all the testing, we reached a number of conclusions:

The first phase of testing showed us that AutoBroad could match the manual performance of the average user, as well as the enjoyable quality necessary to warrant being watched.

The second phase of testing showed us how the manual performance can fail, and how it escalates the mistake to all spectators, which is exactly the prime motivation for AutoBroad to exist, and be developed, in the first place. Despite it all, results were again close, and optimistic, and showed without a doubt that the first experience is the hardest one. And despite it, AutoBroad managed to keep every tester from "failing" more than half the questions, while Manual version did not.

Tests have proven, though, the system is not yet perfect even for this game, we saw, for instance, that its consideration of raid exits needed tweaking. Despite that, it has shown good results. We have to take into consideration, also, that this system is an application, and as with any application, it does not meet its full potential until it goes through a trial of testing and correcting, like beta-testing. These phases of testing would have been that, beta-testing done by users to the application, to derive what could be improved. We were not surprised to find out improvements could be made, though it was optimistic to see how few are necessary, but then again, we did test the system during its alpha stages, with the developers and experts.

While the most important factor for the likability of the Manual broadcasting is the control it offers, the most important factor for the likability of the Automatic broadcasting is exactly lack of control, or better said, lack of responsibility. Not having to worry about whether or not the user is looking at what he should is a big plus, as long as AutoBroad does his job right. Nevertheless, users did enjoy the Manual version the most, shows Phase 1 of testing. It is expected, in a simple game as this with only two players, for the Manual version to be preferred, but the fact it was actually a tie was surprisingly good news.

Also, taking into consideration that no spectator actually controls the spectating of an E-Sports match, all that matters is that testers enjoyed AutoBroad as much as they would any other video-game broadcasting, and that it gave them accurate answers.

Definitely, we feel this was a well worthwhile endeavour, and results show, in summary, that it is worth researching the development of a real-time automatic broadcasting system.

6.2 Future Work

There is great potential for future work here. This document literally births the concept of an automated broadcasting system, and shows that it is possible.

Light weight and functional, AutoBroad can be applied to any video-game. Future work is not only limited to extension, which is to say, applying AutoBroad to player-

based TPS videogame matches that hold more than two players, but it is opened to the exploration of other game genres. How different will it be, for instance, to apply it to a first person shooter, like Counter Strike? Or a Real Time Strategy game, like Starcraft 2?

But mostly, AutoBroad is a concept that covers more areas than we researched and implemented. We remind that we barely touched the surface of the Camera System module. Our vision was to have viewport animation to smoothly change between focuses in a visually appealing manner, all the while allowing more than one stream. Instant replay is another function that belongs in a system like this, as well as easy configuration. Shot Manager has the timing of its priority checks fixed, for example, so why not enable its configuration so the commentator, or another member of the broadcasting crew, can customize it?

And the camera system: why not enable flythroughs and orbits, other points of view that will relay exhilaration with an edgy shot, and stillness with a tactical overview? Why not infuse the system with the many research that has gone into the emotional translation, and appropriate relay, of a scene? Camera work can be heavily worked upon.

New concepts are the building stones of great structures, and we dare say AutoBroad is as new as it gets nowadays: and as much as it is but one stone, it is a cornerstone for Automated Broadcasting.

References: Literature

[ALM03] N. Adami, R. Leonardi, and P. Migliorati. An overview of multi-modal techniques for the characterization of sport programmes. In *In Visual Communications and Image Processing*, pages 1296–1306, University of Italian Switzerland (USI), Lugano, Switzerland, July 2003. VCIP.

[AS06] W. Andreff and S.Szymanski. Handbook on the economics of sport, 2006.

[Bab06] B. Babatunde. The demand for sports broadcasting. In *Handbook on te economics of Sport.*, pages 100–111, 2006.

[BS96] Bob Burke and Frederick Shook. *Sports photography and reporting*, chapter 12. 2nd. Longman Publisher USA, 1996.

[BSvdH08] S. Bakkes, P. Spronck, and J. van den Herik. Rapid adaptation of video game ai. In *Proceedings of the IEEE Symposium on Computational Intelligence in Games*, pages 79–86, Perth, Australia, 2008. IEEE.

[HB] S. Hladky and V. Bulitko. An evaluation of models for predicting opponent positions in first-person shooter video games.

[HO00] N. Halper and P. Olivier. Camplan: A camera planning agent. In *Spring Symposium on Smart Graphic*, pages 92–100. AAAI, 2000.

[HS08] J. Hsieh and C. Sun. Building a player strategy model by analyzing replays of real-time strategy games. In *Proceedings of the International Joint Conference on Neural Networks*, pages 3106–3111, Hong Kong, China, 2008. IEEE.

[Lai01] J. E. Laird. It knows what you're going to do: Adding anticipation to a quakebot. In *Proceedings of the fifth international conference on Autonomous agents*, pages 385–392, Montreal, CA, 2001. Agents.

[MJY09] H. P. Martinez, A. Jhala, and G. N. Yannakakis. Analyzing the impact of camera viewpoint on player psychophysiology. In *Proceedings of the Int. Conf. on Affective Computing and Intelligent Interaction*, Amsterdam, The Netherlands, September 2009. ACII.

[MM09] J. McCoy and M. Mateas. An integrated agent for playing real-time strategy games. In *Proceedings of the AAAI Conferences on Artificial Intelligence.*, pages pp. 1313–1318, Chicago, Illinois, 2009. AAAI Press.

[MOR08] K. Mishra, S. Ontañón, and A. Ram. Situation assessment for plan retrieval in real-time strategy games. In *Proceedings of the European Conference on Case-Based Reasoning.*, pages 355–369, Trier, Germany, 2008. Springer.

[Rea05] Niall Rea. High-level event detection in broadcast sports video. Technical report, Trinity College Dublin, Dublin, Ireland, April 2005.

[SD97] Drucker Steven and Zelter David. Intelligent camera control in a virtual environment. *Graphics Interfaces*, (Graphics Interfaces 97), 1997.

[SMY09] M. Schwartz, H. P. Martinez, and G. N. Yannakakis. Investigating the interplay between camera viewpoints, game information, and challenge. In *Proceedings of Artificial Intelligence and Interactive Digital Entertainment.* AIIDE, AAAI Press, October 2009.

[XXC+01] P. Xu, L. Xie, S. F. Chang, A. Divakaran, A. Vetro, and H. Sun. Algorithms and system for segmentation and structure analysis in soccer video. In *in Proceedings of ICME 2001*, pages 928–931, Tokyo, Japan, August 2001. ICME.

[YA06] Hector Yee and Elie Arabian. Battle cam: A dynamic camera system for real-time strategy games. In *In Game Developers Conference*, San Jose, CA, USA, March 2006. Game Developers Conference. A Presentation Panel.

References: Websites

[1] imdb webpage for the relevant movie hulk.

[2] imdb website for television series 24, May 2011.

[3] Official site for game development engine unreal engine 3, May 2011.

[4] official us english website for game world of warcraft, September 2011.

[5] Official website for competitive e-sport quake live, September 2011.

[6] Official website for competitive e-sports tournament host, gomtv, October 2011.

[7] Official website for competitive e-sports tournament host, intel extreme masters, September 2011.

[8] Official website for competitive e-sports tournament host, world cyber games, September 2011.

[9] official website for game franchise fight night, May 2011.

[10] official website for game franchise tekken, May 2011.

[11] official website for game guilty gear x2 reload, May 2011.

[12] official website for game guilty gear x2 reload, May 2011.

[13] official website for game heroes of newerth., May 2011.

[14] official website for game league of legends., May 2011.

[15] official website for movie scott pilgrimg vs the world, May 2011.

[16] official website for video-game starcraft 2, May 2011.

[17] official website for warcraft 3 modification game defense of the ancients, May 2011.

[18] Steam website about competitive e-sport counter strike 1.6., September 2011.

[19] Temporary site for game project shadow conclave, April 2011.

[20] wiki site about competitive e-sport wow arena, September 2011.

Part I

Appendix

6.3 First Person Shooter

In FPS (first person shooter) games, each player uses a camera attached to the head of the character, to simulate a realistic first person view. The players use a wide array of weapons and, in most cases, map-related assets like power-ups or ammo, which they have to carefully manage.

Quake Live[5], which is usually a one on one death match, though it can host more numbers and even team matches, is a very strong game in terms of competitive importance in the e-sports world. In it, every player starts with a basic weapon; then, laid across the map, are power ups which offer other weapons, health or armor, or even a damage modifier. They regenerate after a certain time of having been picked up.

Counter Strike 1.6[18] features team play and is, we believe, the most important FPS game played in a professional competitive scenario. There are two sides to every match: Terrorist and Counter-terrorist. It is the goal of the terrorist team to plant a bomb on a marked location, and the counter-terrorists must obviously stop them. If any team completely kills every player on the opposing team, that team is the victor. In this game, there is a great deal of tactical positioning on the map, from every player, like in a game of paintball.

The game is played across several rounds, at the beginning of which players can spend money on military gear, money they earned either killing or not dying.

In general, this genre is really popular in the western scene, and thus dominated by the American and European players.

6.4 Third Person Action

Very popular World of Warcraft[4] and like-minded MMOs and other video-games, represent the genre of TPA (Third Person Action) video-games. In it, there is little to no management of anything but what powers/attacks to execute, and the energy that characters spend on using them. Any customization is done prior to a match. Also, while movement control on RTS and TPS games are indirect (the player points with the mouse where he wants the characters or character to move), in these games, the player often directly and explicitly controls the actions of his character entirely, including movement (Example: using the arrow keys). Very player-based, very straight-forward action-based, but we should mention that championships are usually team vs team, in which they fight to slaughter the opponents. Another fact that sets them apart, since there are no strategic points to vanquish and conquer.

But they are still roleplaying games, and hence, each player has a different role in the fight, which is the determining factor to set it apart from FPS games. For example, on a three versus three match, there is usually someone focused on healing, someone focused on dealing damage, and someone focused on crowd-control (has ways to stun, slow and otherwise debilitate opponents.). It can be very tactical, but in no way does the state of the game world, nor the abilities and capacities of the character, vary during the course of a match.

6.5 Beat 'em Ups

Also known as fighting games. Even though some games have implemented more complex matches than one on one, even in complete 2D (like popular game Guilty Gear[11]), championships are often one on one matches. Each player controls a character with a specific set of fighting moves, and they battle each other to a KO. Examples of these are very famous: Street Fighter[12], Tekken[10], Guilty Gear[11] and, to give a straight example of how the game progresses, Fight Night[9], which is literally a simulation of boxing matches.

6.6 Sports Videogames

The genre worthy of mention represents simulations of conventional sports: Golf, soccer, American football, basketball, racing and so forth. Nothing of mention on this topic, they are always as perfect simulations as they possibly can. We do not include fighting sports here due to the camera work in terms of broadcasting, which we relate to the Beat 'em Ups category.

6.7 Image used to teach Phase 2 Testers

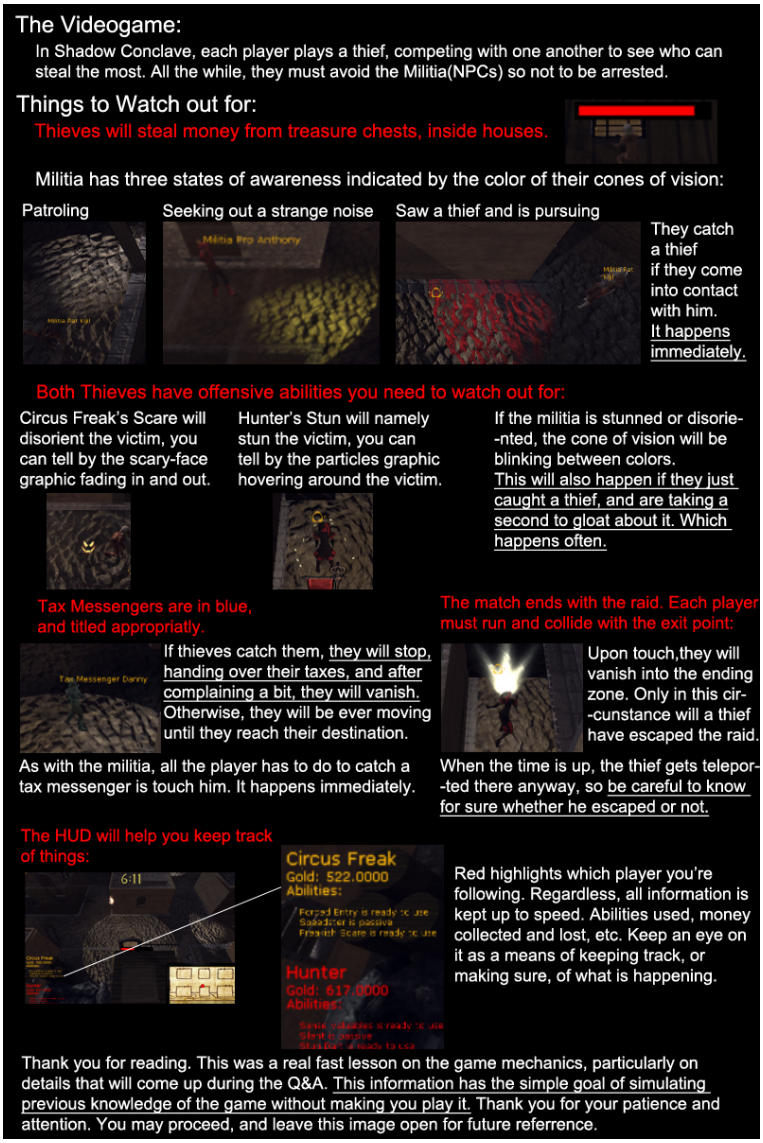


Figure 6.1: Quick Lesson given to Phase 2 Testers.