AutoBroad - A Realtime Broadcasting System for E-Sports

Hugo Miguel Gonçalves Damas

November 25, 2011

Abstract: In this paper, we will propose an automated broadcasting system for multiplayer video game matches that functions in real-time, which we believe is a worthwhile first step to a much more perfected real-time broadcasting system that may come in the future.
Keywords: Intelligent Camera, Situation Assessment, Electronic Sports, Sports Broadcasting,

1 Introduction

One of the rising facets of gaming industry is that of professional gaming. Professional seasons and leagues spawn all across the world, giving the industry a dynamic growth that only the professional sports industry can provide. As this is happening, though, broadcasting techniques employed to cast these matches, multiplayer videogame matches, have shown to be imperfect and inadequate. The way it is currently handled is that two individuals, normally a professional gamer and a professional sports caster, make use of the spectating system, provided by the video-game, to overview the match; they stream one of their feeds to all outlets of distribution and that way viewers and spectators can watch the match.

The problem is this is very limiting. The average broadcasting team of a conventional sport usually has access to, among other things, multiple points of view, numerous cameras filming off-screen to which they can swap to, instant replay, and so on.

AutoBroad is a vision of an automatic 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; not heavy on the global system (video-game); relative global and individual information feedback; instant-replay; multiple viewports for more than one point of interest; animations to transition between more than one point of interest; cinematic or cinematographic presentation.

Certain features certainly involve other domains of research, and could be considered well studied in past works, as well as secondary objectives. This is why we decided to focus on the first six features, leaving the rest(instant-replay onward) to be developed and explored in future work.

We believe 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 the industry at large that is involved with the competitive world of Electronic Sports.

In this paper we will first present a model that can be used as the basis for any architectural model of an automatic real-time broadcasting system, and then how we designed and applied the actual implemented architecture, which took into account a specific video-game, as well as the development platform used to develop it.

The first step towards designing any solution, we believe, is to consider approaches and solutions taken for the same, or similar, goals. With that in mind, we start by delving into related

works which we feel were related to ours.

2 Related Work

Though we believe the concept of automatic broadcasting to be thoroughly new, it still involves several different facets. We divided the kinds of related works across three different areas of investigation.

2.1 *Information Extraction for an Artificial Intelligence*

In terms of video-games, since no one has approached a real-time broadcasting system, we looked at the work developed in the area of *strategy prediction.*

Studies have been promising and advancing, throughout a facet of fronts, offering a myriad of approaches to the problem, that share a few things that differentiate all of them from ours. Some real examples that can be read include "An Evaluation of Models for Predicting Opponent Positions in First-Person Shooter Video Games"[HB08], "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].

They all share some common properties that differ from our work, namely, that 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.

Plus, 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"[HB08]. The rest are tested with scripted players a number of times, in order to derive efficiency against two or three play styles. In summary, they are studies aimed at objectively different goals.

There is one work, though, that we found to be the exception: 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 recognize situations, as in match-state, and from there recognize what strategies are being employed.

The developed AI in question uses the notion of "situation" (game-state), 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*, which all together, define a situation/game-state. 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.

2.2 *Camera Profiling*

We considered the approach of Battle Cam [YA06], the battle camera implemented on Star Wars: Empire At War TM. Using it, Yangli Yee and Elie Arabian presented a camera system to be used for 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.

Looking to only use raw low-level game-state information, 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.
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, *that 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.

2.3 *Sports Broadcasting*

In the domain of studies in Sports Broadcasting, we focused on studies done on highlights extraction from sports videos.
Taking 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, we will see that *they apply shot classification, and use observable heuristics to interpret it*, in order to recognize a match event.
Looking at "Algorithms and system for segmentation and structure analysis in soccer video"[XXC+01], we will find that shot-classification is sport-dependant, and also that the aim of a broadcasting producers are, quoting, to "Convey the global status of the game. Closely follow actions in the field."
Knowing, confirmed by literature of past works, above ones included, 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.

2.4 *Intelligent Camera Control*

- Intelligent Camera Control in a Virtual Environment[SD97] specified a very similar system, but only aimed for exploring virtual worlds, particularly, a museum.
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 wish to have an automatic control over what the camera is going to show rather than, for instance, have our video-game expert pointing out what he wants to show the audience and the camera just moving around based on his choices.
So, in general, we acquired some worthwhile contributions from past works, but this work clearly stands different when compared to any of them, especially in its goal.

3 Heuristics Research

Our final goal is to translate low-level information into high-level information about points of interest, then interpreting that into a kind of broadcasing ontology that says how important it is. So our first step was to describe a list of high-level heuristics employed by current day broadcasting teams, towards shot/event priority.
For that purpose, we observed 56 best-of-three matches from the Starcraft 2[8] championship GSL[3], season 2, held throughout October and November, 2010. We observed about 4 matches of League of Legends[7], from the WCG tournament[5], 2010.
We also observed all world finals in the ESL 4[4], 2010, for WoW Arena[12] , Counter Strike 1.6[10] and Quake Live[2]. 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.
Additionally, we 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 previ-

ously, which included baseball, soccer, cricket, snooker and tennis.
After performing said observation, we retrieved a list of high level heuristic to employ in the system. These were:

- *Conflict* - This heuristic defines the probability of action taking place, presently or in the future, we observed this was, and is, the main concern of any and all broadcasters: *conflict*, which most likely translates to action taking place presently, or soon into the future.

- *Unexpected* - This heuristic defines 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. Once a missed moment was noticed, it took a high priority over anything that was going on and that was not real-time conflict.

- *Tactical* - Decision awareness. In case neither of the aforementioned is applicable, we observed the broadcasting crew just alternating between shots, showing a global view, or partial that, to show what strategies/tactics were being employed through what decisions were being made.

- *Idle* - Finally, this heuristic defines the moments when nothing of interest is happening. We observed the broadcasters roam and hover around while the commentators do their best to not let the interest of the viewers drop too much.

Each of these heuristics will help define how important an event is, by classifying it. But there are also those heuristics we noticed applied in a more global manner. The following three:

- *Interest*: Conflicts involving winning players outweigh those involving losing players. *Survivor*: But within said conflict, the perspective of the losing participant seemed to be preferred. It increases tension as the losing participant may always turn the game around and start winning.

- *Time*: Broadcasters showed a predisposition to change shots, even if nothing different had happened. We suppose the goal is to keep within the attention span of the viewers, avoiding looking at the same thing for too long.

So these are the heuristics we concluded 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. And with that in mind, we defined our architecture.

4 Architecture

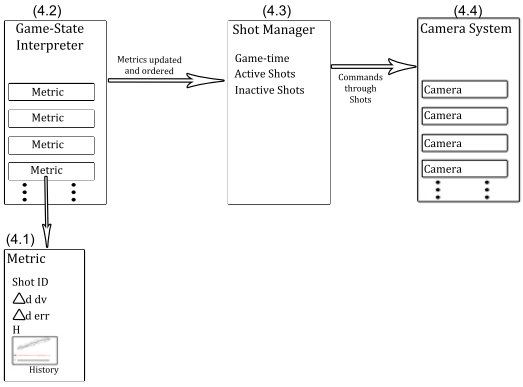

Figure 1: Model of Base Architecture

Figure 1 shows the base architecture proposed to 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 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 what exactly are they evaluating the priority of, in terms of game location. The ending goal is to know what to shoot, as in, knowing 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 proportional 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 unexpectedly. History predicts what the next value will be, the "projected value" which is then compared to the actual value, and if they differ by more than the error margin, it means something unexpected took place, which the Metric takes into account when calculating its final priority value.
Finally, there is the heuristic value, which makes a final contribution to calculating the final priority value of a metric.
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. One of the four described above (Conflict, Unexpected, Tactical and "When-All-Else-Fails", named "Other"). From the most important, Conflict, to the least, Other, the heuristic value is used by the Shot Manager to know how to value metrics against each other.

4.2 Game-State Interpreter

The Game State Interpreter manages the metrics. It marks the pace at which they update, and keeps them ordered. With it we have the first step of Game-State interpreting. 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 data, deriving a situation/match-state(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 and valuing points of interest in the game according to broadcasting heuristics.

4.3 Shot Manager

As stated before, 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 mind, the module adds a deciding factor to the heuristics management, adhering to the two general heuristics: *time* and *interest/survivor* . 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 it uses as a threshold needs to take into consideration what kind of priority values it is dealing with.
We thus have six major guidelines based on

observable heuristics, four of which are already defined by the Game-State Interpreter. Having chosen which shot to show, the Shot Manager contacts the camera system with orders for the appropriate cameras, by passing on to it which shots are active, as well as any extra necessary information.
Lastly, the Shot Manager needs to recognize parallel events. If it had to pick between two, or more, equal priorities, it will have meant it missed one. If this is so, it should save the time of the moment, so it can issue a replay command on the camera that recorded it, at a more appropriate time.

4.4 Camera System

The camera system keeps track of points of interest, actually feeding back to the spectator only when the Shot Manager requests.
It should also show said points of interest, may they be mobile or static, in a way that is visually appealing and not at all confusing, without causing performance issues with the overal system.

5 Implementation

5.1 Shadow Conclave

Now that we have looked at the base architecture for a general automatic broadcasting application, we can apply it to our domain of testing.
We implemented the system on an indie video-game developed in Unreal Development Kit(UDK) [1], named Shadow Conclave [11]. 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. Shadow Conclave is a TPS (Third Person Strategy) game similar to League of Legends[7], Defence of The Ancients[9] and Heroes of Newerth[6], though different in many aspects.
In it, 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, to escape jail. 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 where we tested: Circus Freak and Hunter, which can been seen below, in figure 2.

Figure 2: Characters of Shadow Conclave

Circus Freak is faster than the Hunter but the Hunter is quieter than the Freak. Each of them have two abilities, one which causes harm to an opponent, and one which helps them to steal(open doors faster, for example.)
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(Real Time Strategy) and TPS(Third Person Shooter) games. That was the video-game in which we implemented our system. 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.

5.2 Interpreting Shadow Conclave Game-State

We have defined eight metrics:

- *Competitive Stats* - Informs the Shot Manager of who is winning.
- *Ability Cooldown* - Informs the Shot Manager that this character is about to gain one more avenue of action.
- *Ability Used* - Informs the Shot Manager that an ability has been used.
- *Proximity to Objectives* - Informs the Shot Manager about how close a given player is to treasure chests or raid exit points.
- *Proximity to Foes* - Informs the Shot Manager about how close a given player is, to Standard Militia, Rats, Pro Militia, Tax Messengers or, more importantly, other players.
- *Interaction* - 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* - 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* - Informs the Shot Manager of whether a player has been stunned, disoriented or arrested, and of how long until he recovers.

The *Game-State Interpreter* simply manages the metrics so they are all updated within the cycle of decision of the Shot Manager, spreading the handling of metrics across as many ticks as it can.
Next, since this is a player-based game, every heuristic should be directly related to each character/thief. Likewise, we would have two cameras, one on each player, and then we would pick which one we should show. For this reason, we felt it was more efficient to insert that concept into the functions of the Shot Manager, so instead of going through all the metrics, the Shot Manager processes Shots.
The way we implemented it, the Shot is not only an encapsulation of camera configuration information, thus representing a camera, but is also a module that also keeps track of a point of interest, in our case, a player. Each shot has a list of all metrics that concern its respective player, and a priority value which informs the Shot Manager of how important it is. So the Game-State Interpreter makes sure the metrics are all up to date, and then the Shot Manager looks directly, and only, at each shot, only the inactive ones, to deign which is more important. When the Shot Manager evaluates the metrics within each Shot, it applies the heuristic modifier to each metric, plus the *time* and *survivor* modifiers to the final calculated Shot Priority since they are global heuristics.
After it knows which inactive shot is more important, it checks with how long it has been since it last performed shot switch. Depending on that, and the priority value of the shot, he will perform a switch.
In our case, the Shot Manager performs a check every half a second (its decision cycle). Every

priority value varies within a percentage range (0-100). When a Shot Priority Value is calculated, from each of the priority values belonging to each of its metrics, it is calculated in a way it will reward high priority metrics with a bigger contribution to the total value, but also in that it will not punish a shot for having less "active" metrics (metrics with priority values above 0), balancing the total values despite the fact some have less metrics building them.
The survivor modifier is fixed, but the time modifier fluctuated according to how high the priority is and how much time has gone by. The more time has elapsed, the higher the modifier, the higher the priority, the broader the interval of the modifier; this because its maximum value is fifty percent of the priority value. Again, this is applied to the calculated Shot Priority.
Moving on, every half a second the Shot Manager performs a priority level check, to decide if it should activate his most important inactive shot.

- *Critical Level Check*: Without time restrictions, 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 last activated a 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 last activated a 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 last activated a 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 last activated a 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.

The numerical values shown in these checks are like any other parameters in this system. They have been tuned across a preliminary testing phase, using Shadow Conclave, with that goal in mind. We cannot guarantee they will work optimally for other video-games. But we believe the approach will: which is to make the several checks by looking at both priority and time since a shot was (de)activated.

The *Camera System* is implemented as one camera which switches between the "shots". The camera system is built only as far as is necessary to test everything else. We took this approach in order to test the other modules, and not this extensively researched and developed module.
We really decided to focus all of our effort into building a well functioning game-state interpreting. As it stands, we believe one could pick any player-based Third Person Strategy game like Shadow Conclave, and use the Game-State Interpreter and Shot Manager described here, managing and working with metrics defined specifically for said game, and AutoBroad would be spectating that game just as well as it does ours.

6 Evaluation

6.1 Testing

In order to carry out the testing, we patched the broadcasting system with a secondary version where the viewers could choose which player/shot to see, by pressing the spacebar. It was our goal to compare how AutoBroad

fared against actual users, hoping it would show equal or superior wisdom.
For this purpose, we carried out two phases of testing. Both phases involved having a variety of testers watching a match, casted by one of the versions, and then filling a set of questions. Some of these graded the understanding of the match(functionality of the system), others the experience itself(the quality of the system).
In the first phase, we made sure the testers were unfamiliar with Shadow Conclave, and the system itself, and had them watch two real-time matches, one with each version of the system. They would then answer the questions, and at the second test, they would inform us which one they preferred, and why. In this phase, every test corresponded to a match, and we arranged for a total of 33.
In the second phase of testing, we made sure the testers were familiar with the game in question, but they saw only a recording of the cast of a match. There were two groups, one testing a match casted by the automatic version, and one testing a match casted by the manual version. In this phase, every test corresponded to the same match, and we arranged for 27.

6.2 Results

We evaluated the answers by tester, and by question. In order to better understand the results, we translated the grades into percentage, 100 being the best possible.
In the first phase of testing, the *average grade was between 86.6 percent for Manual mode* and *an average grade between 84.1 percent for Automatic mode*, respectively with a worst grade lying between 75 and 70.8 percent
In total, they differentiate by 2.5 percent, which is approximately equivalent to one question.
In the second phase of testing, the *average grade was between 79.4 percent for Manual mode* and *an average grade between 8.0 percent for Automatic mode*, respectively with a worst grade lying between 64.6 and 68.75 percent.
In total, they differentiated by 0.6 percent. This suggests there is no advantage either way, from this point of view.
In 3, we can see the performance difference, of both phases, with the grades now listed by question: The graphic shows the non absolute

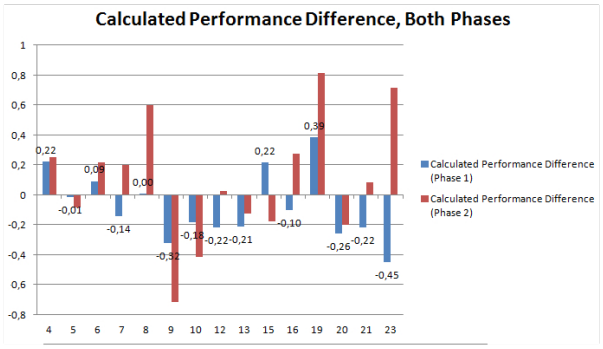

Figure 3: Comparing the overall performance, for each question.

difference between the average real grades (not percentage) of the Manual version and the average real grades of the Automatic version, concerning functionality(the closer to 0, the better). The positive values represent advantage for the Automatic version (it produced more correct answers) while the negative represent advantage for the Manual Version.
This shows which question helped one or the other system, which questions were the hardest and easiest, and more importantly, how little the difference was between the two versions. The tie reflects itsef even from this point of view.

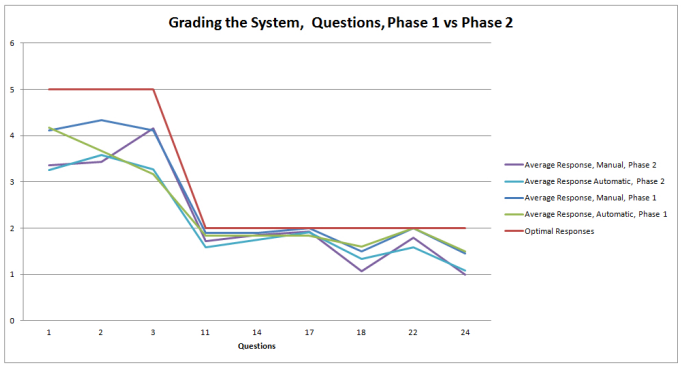

Figure 4: Test Scores, by Question.

Figure 4 shows the answers, by question, to the questions which rated the enjoyability of the system (quality), in both phases. The red straight line shows what would be the optimal value for each question, and the other lines show the values given. We can see all four lines, first and second phase, automatic and manual version, offer values close to optimal, and for that matter, very close to each other. The grades of the second phase, though, are visibly inferior to the values we retrieved during the first phase, which is to be expected as the first phase is represented by second-experience users only while the second phase includes mostly testers that functioned with the system only once. Plus, during the second phase, since testers watched recordings, none of them really had control over any choice.
What is important to retain is that the system was enjoyable enough to warrant watching.

7 Conclusion and Future work

So results have shown AutoBroad could match the manual performance of the average user, as well as the enjoyable quality of a manual system.
We also saw how the user made a mistake while AutoBroad remained faithful to all the details of the Real Match.
While the most important factor for Manual likability is the control it offers, the most important factor for Automatic likability is not having to worry about whether or not the user is looking at what he should. It is expected, though, for the manual version to have the upper hand, especially in a simple game as this, with only two players, but the fact it was actually a tie is evident and proves the system works.
We believe this was a well worthwhile endeavour, and results show, in summary, that it is worth researching the development of an automated broadcasting system.

7.1 Future work

There is great potential for future work here. This documents literally births the concept of an automated broadcasting system, and shows that it is possible.
Light weight and functional, AutoBroad can theoretically be applied to any video-game. Future work is not only limited to matches involving more players, but extending it beyond game genre.
But mostly, AutoBroad is a concept that covers more areas than we researched and implemented, opened for further extesions: viewport animation to smoothly change between points of interest all the while allowing more than one stream, instant replay, a configurable version, a more cinematic behaviour, and so on.
New concepts are the building stones of great structures. 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 multimodal 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.

[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.

[HB08] S. Hladky and V. Bulitko. An evaluation of models for predicting opponent positions in first-person shooter video games. In *Computational Intelligence and Games, 2008. CIG '08. IEEE Symposium On*, pages 39–46, Dept. of Comput. Sci., Univ. of Alberta, Edmonton, AB, December 2008. Computational Intelligence and Games, IEEE.

[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.

[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.

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

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

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

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

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

[6] official website for game heroes of new-erth., May 2011.

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

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

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

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

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

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