BACHELOROPPGAVE:

AR Stadium Games - ARENA: Creating a framework for rapid prototyping of crowd controlled AR games

FORFATTERE:
Arild Jacobsen
Simen Kjærås

DATO:
Vår 2010

Sammendrag av Bacheloroppgaven

Tittel:	AR Stadium Spill - ARENA	Nr: 8
		Dato: Vår 2010
Deltakere:	Arild Jacobsen	
	Simen Kjærås	
Veiledere:	Simon McCallum	
	Sule Yildirim	
Oppdragsgiver:	Simon McCallum	
Kontaktperson:	Arild Jacobsen, jacobsen.arild@gmail.com, 95888163	
Stikkord	AR, Utvidet Virkelighet, video, publikum	
Antall sider: 31	Antall vedlegg: 6	Tilgjengelighet: Åpen

Kort beskrivelse av bacheloroppgaven:
Augmented Reality (AR) handler sentralt om grensesnitt; interaksjon mellom menneske og maskin ved nye metoder. Ideelt vil slike nye interaksjoner være en forbedring; grensesnitt som integreres med virkeligheten i stedet for å tvinge brukere til å stenge den ute:

> "These applications have shown that AR interfaces can enable a person to interact with the real world in ways never before possible."

[1, Section 1] De gamle grensesnittene er imidlertid godt forskanset bak tiår med bevist funksjonalitet og vanebygging. Det kan derfor være problematisk å fange interessen med noe som anses som lettsindig og unødvendig arbeid. Innen spill er imidlertid forventningene at opplevelsen skal være lettsindig, og man forventer å engasjere seg for å lære spillets regler og hvordan det best kan spilles, uten først å måtte bevise noen praktisk hensikt for øvelsen. Dette prosjektet har blitt utført på vegne av Simon McCallum, førsteamanuensis ved Høgskolen i Gjøvik, som tidligere har tatt del i utviklingen av et program for publikumskontrollerte AR-spill. Det prosjektet genererte positiv respons som oppmuntret bidragsyterene til videre arbeid. Koden for programmet viste seg imidlertid å være uhåndterlig og vanskelig å utvide videre. Dermed har hensikten for dette prosjektet i hovedsak vært å generere et håndterlig, utvidbart rammeverk som kan brukes til å gjenskape de spillene som ble demonstrert i det forrige prosjektet, men som også kan brukes til å lage andre, mer eller mindre lignende program og legge til rette for fremtidige utvidelser. Som et sekundært mål, mente vi å evaluere hurtig prototypeproduksjon av spill som en metode for å utføre eksperiment, ved å implementere et spill ved bruk av det genererte rammeverket, og måle reaksjoner fra test-publikum. Tertiært ønsket vi å utvide den grunnleggende funksjonen i rammeverket, dersom tidsrammen tillot det, hvilket ville legge til rette for produksjonen av mer sofistikerte spill.

Summary of Graduate Project

Title:	AR Stadium Games - ARENA: Creating a framework for rapid prototyping of crowd controlled AR games	Nr: 8
		Date: Vår 2010
Participants:	Arild Jacobsen	
	Simen Kjærås	
Supervisor:	Simon McCallum	
	Sule Yildirim	
Employer:	Simon McCallum	
Contact person:	Arild Jacobsen, jacobsen.arild@gmail.com, 95888163	
Keywords	AR, Augmented Reality, video, crowd	
Pages: 31	Appendixes: 6	Availability: Open

Short description of the main project:
Augmented Reality (AR) is in essence about interfaces; allowing humans to interact with computers throught new means. Ideally, these new interfaces improve upon what has gone before, integrating with reality, rather than supplanting it.

> These applications have shown that AR interfaces can enable a person to interact with the real world in ways never before possible.

[1, Section 1] However, the old interfaces are firmly entrenched, and have worked for decades; it can be problematic to get people interested in something that seems like frivolous, unnecessary work. Games however, are expected to be frivolous and require the player to engage and exert themselves to learn the rules of the game and how best to play it, without having to first prove a practical purpose. This project has been completed on behalf of Simon McCallum, Senior lecturer at Gjøvik University College, who has previously taken part in developing a program for crowd controlled AR gaming. This generated positive responses, which left the contributors encouraged to perform further experiments. However, the codebase for the program proved unwieldy and difficult to extend. The purpose of this project was primarily to create a maintainable, extensible framework which could be used to recreate the games our sponsor had worked on previously, but which could also be used to create other, more or less similar programs and accomodate future extensions of the framework. Our secondary intent was to evaluate rapid game prototyping as a viable method for experiments, by implementing a game using the framework created and gauging the reactions of one or more test crowds. Tertiary was our intent towards using the extensibility to expand upon the base functionality if time allowed, allowing for more sophisticated games to be created.

AR Stadium Games - ARENA: Creating a framework for rapid prototyping of crowd controlled AR games

Arild Jacobsen
Simen Kjærås

2010/04/20

1 Preface

We would like to extend our thanks to our sponsor and supervisor Simon McCallum, as well as the various people we have pestered into guinea-pig duty during the development.

Contents

1 Preface . . . **1**
Contents . . . **3**
2 Introduction . . . **5**
2.1 Project organization . . . 6
2.2 Report outline . . . 7
3 Design . . . **9**
3.1 Specification . . . 9
3.2 Preliminary design . . . 9
3.3 Game design . . . 10
3.4 Implementation . . . 13
3.5 Testing . . . 17
4 Analysis . . . **19**
4.1 Discussion . . . 19
4.1.1 Results . . . 19
4.1.2 Choices . . . 22
4.2 Shortcomings . . . 24
4.3 Further development . . . 24
4.4 Group evaluation . . . 26
4.4.1 Organization . . . 26
4.4.2 Distribution of work . . . 26
4.4.3 Subjective experience of the Bachelor thesis . . . 26
5 Conclusion . . . **27**
5.1 Final summary . . . 27
Bibliography . . . **29**
A Definitions . . . **31**
B Schedule, Gantt-diagrams, milestones . . . **33**
C Log . . . **37**
C.1 Pre-sprint Scrum . . . 37
C.1.1 21.01. . . . 37
C.2 First Sprint . . . 37
C.2.1 22.01. . . . 37
C.2.2 25.01. . . . 38
C.2.3 26.01. . . . 38
C.2.4 27.01. . . . 39
C.2.5 28.01 . . . 40
C.2.6 04.02 . . . 40

C.3 Second Sprint . . . 40
C.3.1 05.02 . . . 41
C.3.2 08.02 . . . 41
C.3.3 0 . . . 41
C.3.4 10.02 . . . 41
C.3.5 11.02 . . . 42
C.3.6 12.02 . . . 42
C.3.7 16.02 . . . 43
C.3.8 17.02 . . . 43
C.3.9 18.02 . . . 43
C.4 Third sprint . . . 43
C.4.1 19.02 . . . 44
C.4.2 22.02 . . . 44
C.4.3 23.02 . . . 44
C.4.4 24.02 . . . 45
C.4.5 25.02 . . . 45
C.4.6 26.02 . . . 45
C.4.7 01.03 . . . 46
C.4.8 02.03 . . . 46
C.4.9 03.03 . . . 46
C.4.10 04.03 . . . 46
C.4.11 05.03 . . . 46
C.5 Fourth sprint . . . 46
C.5.1 08.03 . . . 47
C.5.2 09.03 . . . 47
C.5.3 10.03 . . . 47
C.5.4 11.03 . . . 48
C.5.5 12.03 . . . 48
C.5.6 15.03 . . . 48
C.5.7 16.03 . . . 48
C.5.8 17.03 . . . 49
C.5.9 18.03 . . . 49
C.6 Fifth sprint . . . 49
C.6.1 19.03 . . . 50
C.6.2 22.03 . . . 50
C.6.3 23.03 . . . 50
C.6.4 24.03 . . . 50
C.6.5 25.03 . . . 51
C.6.6 26.03 . . . 51
C.6.7 29.03 . . . 51
C.6.8 30.03 . . . 51
C.6.9 31.03 . . . 52

C.6.10 01.04 . . . 52
C.6.11 06.04 . . . 52
C.6.12 07.04 . . . 52
C.6.13 08.04 . . . 53
C.6.14 09.04 . . . 53
C.7 Sixth sprint . . . 53
C.7.1 12.04 . . . 53
C.7.2 13.04 . . . 54
C.7.3 14.04 . . . 54
C.7.4 15.04 . . . 54
C.7.5 16.04 . . . 54
C.7.6 19.04 . . . 54
C.8 Seventh sprint . . . 55
C.8.1 20.04 . . . 55
C.8.2 21.04 . . . 55
C.8.3 22.04 . . . 56
C.8.4 23.04 . . . 56
C.8.5 26.04 . . . 57
C.8.6 27.04 . . . 57
C.8.7 28.04 . . . 57
C.8.8 29.04 . . . 58
C.8.9 30.04 . . . 58
C.8.10 03.05 . . . 58
C.8.11 04.05 . . . 58
C.9 Eight sprint . . . 58
D Code Samples . . . 59
E Coding standards . . . 65
E.1 General . . . 65
E.2 Classes and structs . . . 66
E.3 File names . . . 68
F Contents of the DVD . . . 69

2 Introduction

Although Augmented Reality as a concept has been around for decades, and the ideas from which it arose can be traced back however far you would care to argue for, the use of AR as a serious tool is still in its infancy. Although it is difficult to find a definitive history of AR technology, it is even more difficult to deny that the use of and interest in AR has grown considerably the last decade, and in the past few years has been noticed by a much wider audience.

The concept of audience-interactive entertainment can also be traced into the mists of history. Even the meaning relevant to this project, using digital equipment to facilitate entertainment that an audience can directly control or at least affect, goes back surprisingly far: Maynes-Aminzade, Pausch and Seitz mentions that there had been little development in the field since Cinematrix launched in 1991 [2, Section 1], then go on to attempt to improve on the idea. About a decade later, Sieber, McCallum and Wyvill set out to prove the concept again [3]. Given that at least two papers before us as well as Cinematrix, a decade earlier again, have proven that it is entirely possible to provide this manner of crowd-controlled entertainment, our purpose is not to prove this again, but rather to provide the means for others to continue that work.

There are several reasons that led us to decide on this subject as the focus for our project. For one, the topic of AR is of great interest to us, not only because we believe it may well grow to be an important element in future IT-applications, but because the field is full of interesting challenges. In addition to the various aspects of image analysis inherent in video-based AR, the most common form of AR, there is the challenge of designing applications quite unlike what is considered normal.

Quite apart from the challenges, we were also genuinely interested in the *BallBouncer* project, having experienced some of the games personally, and as such had an interest in seeing it developed further. Not only the BallBouncer itself, but the general concept of crowd-controlled games is one which we both wish to see further work on and would be interested in being personally involved in. Both AR and Crowd Games contend with established paradigms, and as such, a program which facilitates experimentation with one or both concepts seems a worthy project.

Starner et al. claims that games are an excellent way of prototyping and testing new interaction technologies[4, Section 1], because games are about play, and play is all about experimenting and learning. The Nintendo Wii is an excellent example of how willing people are to experiment if it is in the name of entertainment. As both Sony and Microsoft develop their own alternate interaction technologies, it does not seem presumptuous to claim that the atmosphere is ready for new modes of play. If people are willing to accept new technology in a game and experiment, we are provided with excellent data about how it is perceived and used.

Our project had the benefit of being rather clearly defined from the start, at least as far as the desired results. Our sponsor had a working program [3], and he wanted the functionality duplicated, but the code that achieved said functionality improved upon. Our task then was to replicate the results of a known program while keeping scrupulously to good coding standards

for maintainability and extensibility.

However, there were three target audiences to consider in our task. First there is the sponsor, for whom we were to produce a good codebase to build future extensions to the project upon. Second there is the audiences who will use the games or other applications produced from this framework; unless they are satisfied, the purpose falls flat. Thirdly, as this is an academic project, we should produce some theoretical insight into the subject matter. Thus, while the program resulting from this work is an important part of the project, we also had to continually evaluate the quality of the code we produced from the standpoint of a future developer, and lastly consider the purpose and role such a program might serve. In short, we had to balance what we did with the how and the why of it.

Much of this project was a new experience for both of us. Neither had used Linux extensively before, and both had only a passing knowledge of Eclipse. We had expected that debugging in Eclipse would be easier than it turned out.

We used a sizeable collection of libraries, most of which were unfamiliar to us, and as such required time spent to familiarize ourselves with them. We had both enountered ODE earlier, and thus had a pretty good idea what use it could be put to. OSG was something completely new for us both, and more than once we found that the functionality we wanted, and had started writing, was already present there. The video libraries we used, ARToolKit and FFmpeg, were complex and underdocumented, while POSIX threads were not hard to understand, but their use felt archaic. Boost, however was a breath of fresh air. Well-documented, easy to understand, and producing easily readable and maintainable code.

2.1 Project organization

During this project, we employed an Agile software development methodology, called Scrum. The choice of methodology was based largely on the small size of our team, but also on the expectation that given the size and complexity of the project, we will meet with several unforeseen issues. Unfortunately, due to the fact that we only had two programmers, and could not spare one to act in a purely administratory role, we had to forgo having a Scrum Master, which would have been the norm when using Scrum. We also deviated from the two-week Sprint cycles at times, at times due to planned exceptions, such as immediately following a milestone, at other times due to unforeseen issues such as illness.

Apart from the two team members, there were two members of faculty who were involved in our project. Sule Yildirim initially served as supervisor, and we intended her to partially fill the vacant role of SCRUM master. Simon McCallum served as sponsor, and after a period of some confusion, came to fill the role of project supervisor as well.

We made use of a Subversion revision control system, both to coordinate our coding efforts and to provide a backup of the current and previous versions of the program. We adhered strictly to the principle of only comitting working builds to the repository, so as not to impede the progress of the other team member, and so that we would have a working version available for demonstration at all times. We also commented all code according to predetermined standards, while keeping the code as self-explanatory as possible.

We held our Scrum-meetings every day at 10:00, to review the progress during the previous

session and to plan what needed to be addressed during the coming session. The Sprint meetings were held. At these meetings we reviewed whether we'd accomplished what was planned for the preceding two-week Sprint cycle, discussed how we could adjust our work process to better accomplish our goals in the next cycle and planned the work to be done during the following Sprint cycle. We also created a Gantt-diagram for each Sprint, illustrating our intended course for the following cycle. No milestones were planned within these cycles. Instead, milestones were set in the overall plan, and provided goals to work towards.

We maintained a log for the Scrum and Sprint meetings, which was accessible to the project coordinator and sponsor as well as each team member. The log has been included in appendix C. We intended for the project supervisor to attend the Sprint meetings which would then serve simultaneously as status meetings; however, due to some confusion as to her role, Sule Yildirim was phased out during the project, and eventually Simon McCallum was phased in, to serve as supervisor as well as sponsor. Because of this, a large part of our development lacked both a supervisor and a Scrum Master, given that we had intended for the supervisor to fill that role. Nonetheless, we kept to the principle of making the larger decisions during these Sprint meetings, allowing for detail decisions to be made during the daily work.

2.2 Report outline

This report consists of four main parts following this introduction. First, we have a section for Design, where we elaborate on the the constraints shaping our design, the choices made during the preliminary phase and the reasoning behind them, which guided our continually developing design throughout the project. Second comes Implementation, which details the process of creating the program; how we went about it, tools used, problems encountered and changes made during this phase, as well as the reasons and reasoning respectively responsible. Following that is the chapter Analysis, where we discuss what we did and what we did not, what we could have done and what we should have done, as well as what could still be done. The main topic is the results and our deliberations about those. We will also delve into such failures as arose during the process, as well as considering possible further developments and projects based on our work. And of course, here as well, we will delve into causes. Closing the chapter will be our evaluation of ourselves, our organisational efforts and our experience of the enterprise. Finally we have Conclusion, which (quite concisely) covers the conclusions we draw from our work.

There is a plethora of technical terms and acronyms used in this report, all of which are listed and explained in appendix A.

3 Design

The design of a computer program can be, as those who have any experience with the process will know, a highly complex process. There is software compatibility, hardware compatibility, resource requirements as well as myriad usability requirements. However, the design of AR applications, as well as Crowd Games, has not been as extensively analyzed and the design of these applications is thus guided largely by technical considerations. Furthermore, we primarily produced a framework, which would be used only by programmers to create applications. Those applications might need a degree of user-friendliness, but our main consideration was producing code which would ease the work of such programmers, while still keeping to our technical constraints.

3.1 Specification

Our specification was somewhat out of the ordinary in that we had a working program, whose functionality we sought to ape, without there necessarily being any similarity in the code producing the results. There were however some additional specifications to consider, and these were arrived at through meetings with our sponsor. As mentioned in Section 2.1, we were working under an AGILE methodology, and as a result no final design document was produced; AGILE methods rely on continual communication more than rigid design documents. As such, the main specifications were set and guided our rough design, and the details were continually evaluated during the work-process. We had contact with our sponsor throughout the project period, and the design was reevaluated multiple times by discussing current functionality and what changes or additions should be done, on occasion resulting in significant alterations.

3.2 Preliminary design

Even though our design was intended to be fluid to allow for changes during the project, there were certain technical limitations we had to take into account, as well as some restrictions set by our sponsor at the inception of the project. Notably the concept of a flexible framework: the main purpose of the project was after all not to replicate previous efforts, but to create an improved replacement. A few guiding principles were established in the preliminary design phases, which would guide all decisions made at later stages:

- Easy to extend
  - The base framework should provide the basis for further work. This guided the decision to keep the code highly modular.
- Easy to maintain
  - The intent for the project is to produce code that will be used by other programmers to create applications. As such, it must be easy to read and understand what the code

does and why it does it. This is often referred to as maintainability in programming; the ease with which a programmer who was not involved in the creation of the code can understand and update (or indeed change) it. This guided our creation of strict coding and commenting standards (see appendix E).

- Use FOSS
  - By restricting our use of libraries to Free/Open-Source Software, any subsequent project based on extending the framework will be unhindered by legal issues. This guided, quite logically, our choice of libraries in creating the framework.
- Natural interaction
  - One of the main tenets for good AR applications is that the interaction between real objects (e.g. the players) and virtual objects should seem natural and obvious to the user [2, Section 5.1]. This was central when during the phases where we wrote the code for acquiring input from a video stream and modelling the behaviour of the virtual objects.

Some additional technical restrictions were also imposed, which constrained, rather than guided our design choices. First is the environment in which the application would run. Ideally, the application should run comfortably on an unexceptional laptop computer, using a Linux OS. The setup should not require more in the way of additional equipment than a fairly average camera, as could be connected via USB or FireWire. This was expanded upon during later meetings to require that the program would instead make use of any generic videostream, regardless of its source, amonsgt other things opening up for the possibility to have video streamed from a remote location over a network.

The resolution of the video stream was also a relevant concern, as the resolution is directly related to the amount of processing power necessary to perform the necessary operations. However, nor could the program perform its job well given too coarse a resolution, since the information required to make an application usable could then turn out to be too fine-grained to detect. The required resolution varies depending on the application and the physical proportions where the application is to be used. The further away from the camera an audience is, the higher the lower limit of resolution needed to detect their movement, whereas the available computing power, which varies depending on the machine used to run the application, sets the upper limit on the resolution that can be computed at an acceptable rate.

After considering this for some time, the conclusion was that the only sensible approach would be to define test parameters which we would hold to during the production; guided in part by the experience from our sponsor from *BallBouncer* [3], and in part by the available hardware, the requirements were that the laptops we used for working on the project should be able to run the program at a rate of 30 frames per second at the least, using a video stream resolution of 800 by 600 pixels.

3.3 Game design

During the course of the project we also had occasion to design a game to be implemented using our framework. Though the time-constraints foreshortened this design phase considerably, we

have arrived at some guiding principles in general for creating such games, which we attempted to employ in our demonstration game. Though it is certainly possible to create a game using this framework which follows an entirely different approach to its design, we based our design approach on the experience gathered from the previous iteration [3], and other examples we could find that seemed relevant [2], [5].

Despite the game's primary role as a test of the framework, we wanted it to live up to certain standards. The quality of the framework is necessarily reflected in the quality of games it can produce, and unless the games produced could competently demonstrate special features of AR games, we would not prove that we had achieved the initial purpose of the project. The game we set out to create was intended to exemplify some of the possible strengths of an AR application. Three main criteria were defined for the game:

1. Casual Play should be voluntary; since the game is intended to be played by a random crowd of people, if some members of an audience do not want to play, they should not feel uncomfortable about it, nor should they be an obstruction to other players enjoying the game.
2. Simple Keeping the game simple was considered of paramount importance, for two reasons. One is that AR augments reality, which is already full of information we relate to, therefore it is sensible to keep any augmentation either simple, integrated or both. The second reason is that the intended use of the game was to have test audiences with little or no previous experience, at a moment's notice, which requires that both interactions and goals can be grasped quickly and easily.
   - As mentioned in 3.2 above, good AR interaction is natural and intuitive. This would serve our purposes twofold by assisting test audiences in learning to play the game and in demonstrating the qualities outlined in *BallBouncer* [3, Section 2].
   - The purpose of keeping the game simple to grasp was not just to minimize frustration for those who would play regardless; we also achieved a very low threshold for joining in. Even those who might not otherwise be interested in playing might be enticed; thus providing more feedback, as well as feedback from a wider range of the audience (i.e. those less interested in new technology).
3. Engaging While the other two criteria deal with avoiding obstructions of the game experience, there must of course also be an engaging game experience for anyone to take an interest. We needed to define some simple game mechanic that would entertain a crowd at least for a middling period of time.

Taking our cue from the original *BallBouncer*, we found that Sieber, McCallum and Wyvill make several observations regarding the efficacy of various gametypes [3, Section 4], noting that the competetive bomb game seemed to produce the greatest activity in the audience. This meshed well with the results of Maynes-Aminzade, Pausch and Seitz [2, Section 5.3]:

> With the beach ball, the lottery effect ("I might be next!") and the cheering or booing of one another fully engages all of the members of the audience

. Based on this, we decided to build a competetive game, and given that our main purpose was to prove we could emulate the games from *BallBouncer*, our game design was based on the bomb

game mentioned there [3, Section 2].

Jegers [5, Section 4.4] states

> "Pervasive games should enable the players to easily pick up game play in a constantly ongoing game and quickly get a picture of the current status in the game world"

. He reasons that this is necessary due to the time/place independent game play that is a hallmark of pervasive gaming making it prohibitively difficult to allow players to pause or save a game. Though the concept of Pervasive games is a different topic than AR games, they do share a common field in the design of unusual game mechanics due to their unusual nature. While our game was not to be place independent (each game session takes place in a static location), it was intended to be partially time independent. For it to be possible for players to walk into a game in progress, they necessarily need to be able to pick up on the current state quickly. Additionally, our intent to create a game which required no experience (2), required that the interactions and goals are equally simple to grasp quickly.

Thus, the ideal of casual play (1) is largely supported by the simplicity of design, in making it simple enough for players who walk into an already active game session to understand and join in. However, to allow for those members of an audience who do not wish to play, we rely largely on the elegant solution of teams. Not only is it an entirely sensible and obvious way to set up a game in these circumstances, it also means that the players are teams, not individuals. Each player is not important in him- or herself, importance is achieved as a part of a group, and as such the absence or lack of involvement of some audience members does not detract greatly from the overall team experience.

3.4 Implementation

Our work was carried out on everyday laptops, running **Ubuntu** 9.04. The platform was chosen mostly because our client suggested it, but it was also interesting to us as we had not been using Linux much in the past. Linux offered a comprehensive collection of open-source tools for software development, a fact that was not overlooked. There were some hiccups, the most major of which was a broken kernel update on one computer, forcing a reinstall of the entire system.

In the bewildering ecosystem of programming tools for Linux, we chose to use **Eclipse** as our IDE. Eclipse is a powerful tool with most features we know from other IDEs, like Visual Studio. It is a cross-platform system, written in Java. We found it mostly pleasant to work with, with some unpolished areas, the most glaring of which was debugger integration. Or perhaps more correct, the lack of such. We were not able to get **GDB**, the GNU Debugger, to work with Eclipse, and this led to some cursing and unrest. After searching for solutions for a while, we decided we could live with the problem, and moved on. While breakpoints are handy, with some ingenuity one can do impressive things using only `printf` for debugging. We also tried using **DDD**, the Data Display Debugger. Perhaps with more experience in its use, it would have proven helpful for our task.

With GNU/Linux comes the impressive suite known as GCC - The GNU Gompiler Collection. It is a collection of Free/Open-Source compilers for many languages, the most interesting of which, for us, was **G++**, the GNU C++ compiler. It has proven to be a high-quality compiler and one of the pieces of our project that caused us no problems.

Our first task was to gather the required libraries and in some cases compile them to static library files. This wasted quite some time, as not all libraries wanted to compile. We found that several were easily accessible from Synaptic, the package manager in Ubuntu.

We set out with the goal of rewriting the original codebase piece by piece. As it turned out, the time required to understand the old code, in addition to the work of actually doing the aforementioned, would have taken significantly more time than implementing everything from scratch, as well as giving a less satisfying end product, so the latter approach was favored instead. We had greatly underestimated the helpfulness of the chosen libraries, and thus reached our first milestone way ahead of time. The other effect of using unfamiliar libraries was that most our carefully laid plans were thrown wayside, as they were incompatible with the way these libraries worked, and the approach of starting from scratch rather than refactoring old and unknown code.

For the graphics and window setup, we used OpenSceneGraph (OSG), an open-source scengraph available for free. For the physics simulation, Open Dynamics Engine (ODE), an open-source physics engine, was used. We created a simple class to unify OSG and ODE (Open Dynamics Engine). At this point, we encountered the problem that ODE by default is compiled with double-precision calculations, while OSG uses single-precision. We tried recompiling ODE to fit our needs, but this turned out to be more trouble than it was worth. We instead added some utility functions that take care of the conversions necessary for interoperability between the two libraries.D

The project called for a graphical user interface. The nature of the application means its use would be output only - displaying scores, messages and possible game-enhancing overlays. Work

was undertaken to provide classes that could display the necessary information, and it was soon after found that this functionality was already present in OSG's impressive hierarchy of classes. However, certain functionality was desired to be easier to access, and classes facilitating that were created.

It was early decided that the physical layout of the world in which our game was to take place, would be defined by settings stored in an XML file. Libxml was used for this purposeD, with utility functions added for ease of access to the kind of information we would use.D libxml turned out to be very sensitive to misformed XPath queries, leading to a number of inexplicable bugs. These were luckily squashed with time.

The original plan called for using ARToolKit for video feeds. This seemed like a promising idea until we discovered that ARToolKit stubbornly refused to work. No example, nor other piece of code with ARToolKit, was able to display anything on the screen. After numerous recompiles and fiddling with settings, we abandonded this path, and chose instead to try and use FFmpeg. FFmpeg was slightly more cooperative, and after only a few days, we could play video files. Webcam support however, gleamed in the far distance, and would take a long time coming. As it turned out, supporting both webcams and video files seamlessly was impossible. Instead, the choice has been moved to the settings file, where it must be explicitly defined. Luckily, most applications would only ever use webcams, so configuration would mostly not be necessary. Following this, problems surfaced around the system for creating difference images from the last two frames. The problems were tracked down to some piece of shared memory, which was turned less shared to ameliorate the problem. At this point, finally, things worked as they should. All in all, almost a month was spent tracking down problems with the video feed, making it the single largest time sink in the project.

The application might take input from several sources, none of which necessarily is in sync with the others. This called for a multi-threaded approach. In hindsight, a message-passing architecture might have been more fitting than the shared-memory model we chose. We chose to use POSIX threads for this purpose, then at a later point replaced them with boost::thread, which is a cross-platform concurrency library. The reason for this change was a surge of problems believed to be related to threading. In the end we also found that boost::thread made for cleaner, more understandable code, and thus chose to keep the new system.

As the design panned out, it proved necessary to organize the threading better, and a thread manager was created. It takes care to kill all threads when they are no longer needed, and makes sure threads are around as long as other threads need them to be. It also proved useful for debugging, as output could then refer to threads by their function, not just their OS-given id.

The application's ability to use several input sources was designed for three reasons:

First, this could give the application a further multiplayer dimension, allowing one to stream video from a different location and thus play against people not present.

Second, one might choose to use a low-resolution camera for the movement source, to ease processing, and a higher-resolution camera for displayed data. This would lower the processing power needed, while increasing visual quality. The loss in movement precision is negligible.

Third, using several input sources for movement would make more complex movement detection possible, like detecting the depth at which movement occurs. This would then use the

projected position of game objects to identify points of interest, and then triangulate to find if movement detected at the appropriate location in one source is the same movement as detected by another source.

Before we had camera input working properly, we were able to read video files from the harddrive and display those. This allowed us to create the class responsible for processing the received frames. The original plan was to perform the processing on the GPU in a separate OpenGL thread. This approach failed, and a software solution was chosen in its stead. Sadly, using a single thread for such a parallel task is very inefficient compared to using the GPU, and it is an unpleasant bottleneck, as the GPU solution should be able to perform this task for virtually any resolution images, and our solution stutters at a resolution of 640x480 pixels when we attempted to go beyond the basic processing functionality, such as processing the created difference images to provide an data for a more sophisticated movement detection. In writing this functionality, it was also obvious that the compiler be unfit for optimizing such a piece of code, and lower-level abstractions had to be used.D

The application is higly reliant on its ability to detect and react to movement. This first piece is performed by going through the received image pixel by pixel and comparing it to the previous received frame. The pixel values of one is subtracted from the other, and the absolute value is stored in a new image. This gives us an image of what changed between the two frames, and most of this difference would be caused by movement. Changing light conditions could also severely affect this, but the code does not take this into account.

Once the difference has been calculated, each object in the world is tested - first for adjacency to the 'floor' of the simulation, which represents the seating of the audience. Next, if the first test passed, for movement in its covered screen section. The former is a simple distance test, the latter divides the bounding square of the covered area into four smaller squares, and sums the differences in each of these to see which has more differences, thus more movement. The average location of this movement is then chosen as the originating point for a force affecting the object.

Various approaches for getting the screen coordinates of physical objects were tested, and we settled on using matrix math in much the same way OpenGL and other rendering systems do. This uses OSG's view and camera manipulator classes to calculate the screen coordinates.

Other approaches for the calculation of force were considered, amongst those the calculation of center of movement for the bounding square, equally bounding circle, and other subdivisions of the bounding square, like 3x3 instead of 2x2. In the case of different subdivisions, and even the subdivision chosen, movement could have been calculated from more than one subsection. Our approach was chosen for its simplicity.

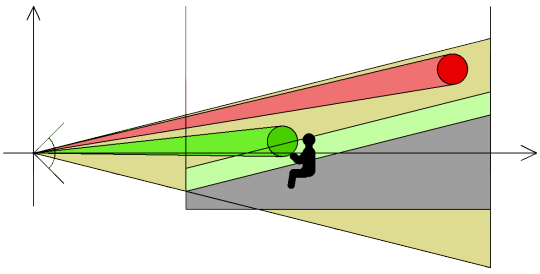

Figure 1: The projection of objects onto the screen.

Several of the objects in the project made sense to only have one of. In these cases, we

used the singleton pattern. The classes for which this made the most sense were clStadium (the physical representation of the locale) and clWorld (the simulation world, i.e. its physical rules, container of the objects in it, and functions that relate to certain other monolithical parts of the program).D

Another design pattern implemented is the monostate pattern. This is used for clThreadManifold. The monostate pattern is a close cousin to the singleton, except it uses no constructor. Because of this, the monostate pattern is to be used only when the object's creation should not change any state.D

In the cases where it was possible and logical, our classes inherit from OSG's own, related classes. Our clObject is an osg::Transform, to represent the fact that it moves with ODE's physical representation of the same object.D

3.5 Testing

Our application did not carry large amounts of unit tests, nor a battery of other tests. Instead, we relied on hands-on testing and painstakingly thought-through code. Due to the agile methodology chosen, we went to lengths to ensure the current version of the application in the repository would be as working as we could make it. That included compiling with no problems, and no regressions. Only once was this requirement broken, and it was fixed five minutes later.

Due to the restriction of always having a running version, we were able to catch many bugs long before they became a serious problem. Due to the graphical nature of the application, most errors could be spotted by simple visual inspection of the monitor while the application was running. Most errors found and fixed in this way were sign errors and its ilk, while more insidious errors could be identified after fastidious inspection and numerous smaller tests under controlled circumstances to provoke behavior thought to have been spotted at an earlier point.

An example of the latter kind was a bug in which the movement detection moved objects in the opposite depthward direction, giving the objects a behavior seemingly innocuous, but still wrong. The specific error was that the force exerted on the object was directed from the center of the active area toward its edge, rather than the other way around, and was fixed by inverting the vector's x and y components in the local coordinate system.

Due to the lack of proper debugger integration in our chosen IDE, `printf` was used for debuggingD, alongside stack traces for `SIGSEGV` - segmentation faults. The latter are not really supported on Linux platforms, as the program is in an undetermined state if such a signal has been sent. Disallowed activities after a segfault include `printf`, `new`, `malloc`, `delete`, `free`, and any code that is not re-entrant. This left us with `write`, which is good enough for our purposes.

In addition to simple visual inspection tests, we routinely tested the interaction on people around us, to make sure the program worked the way it should. This helped us locate several errors and quickly deduce their solutions. A few larger-scale tests were performed with more than a handful of people invited. All in all, this approach worked well for the rooting out of errors in movement detection and related areas, and provided good feedback regarding user experience and possible alterations to improve it.

Sadly, certain errors were not easily testable within the system chosen, multi-threading issues chief among them. Unit testing could have been used more rigorously, and our system did not help us pay enough attention to side effects.

On the topic of multi-threading, it should be mentioned that a message-passing architecture would probably have simplified testing of multi-threading related issues, as it would reduce the number of side effects immensely. This was not done due to constraints of knowledge and time. Implementation of such a system may be worth investigating in the future.

As our work was based on that of others[3], our goal was not to test the merit of the augmented reality ball-bouncing game, but instead that the application be in itself a flexible testing framework for augmented reality crowd games, allowing one to quickly set up and experiment with new game ideas. A simplified solution for creation of such games would make future researchers in the field more productive and reduce their time lost to menial work. As such, the most important test of our work was that we should be able to use our framework to produce a game similar to those demonstrated by BallBouncer [3], which we performed by more or less

replicating one of the mentioned games and gauging the response of an actual audience during our presentation at the Hamar Game Challenge.

4 Analysis

4.1 Discussion

4.1.1 Results

In this project, we have created a basic framework that supports the creation of simple AR games. What we set out to produce was primarily a framework which allowed for the quick implementation of many different ideas based on the central concept of movement in a video affecting virtual objects. An important goal for was that it could reproduce the games shown in *BallBouncer* [3], but it was just as important that we not be limited to those, or the project would have achieved nothing but recreating an already existing program. We were also interested to see how users reacted to using the program. Though *BallBouncer* has already found audiences to be enthusiastic [3, section 5], it was important for us that our program could engender the same kind of response. In fact, the somewhat crude nature of our game example might be seen as a benefit, in that we were interested in determining the viability of using our framework to quickly prototype an idea, be it for a gameplay idea or to test an interaction mechanic, and our test can be taken as proof, however limited, that a basic prototype is not a hindrance in getting an audience engaged.

When all is said and done, at the end of our project period, we have created a functioning framework, and while it is not as complete as ideal, it has proven that it is up to the job of prototyping an AR game in a short time. We presented a game at the Hamar Game Challenge which was created by the two team members in less than a week, and we claim it proved both playable and enjoyable. Prior to our presentation, the audience were not aware they would be playing a game, they were simply informed they would have the option to try the game, and were given a brief half-minute introduction to the rules before we started the game. Following our demonstration, we asked several simple yes-no questions about their experience, assessing the response to each question by asking those who agreed to raise their hands and then having the camera used during the demonstration provide a screen-capture of the audience after each question.

As can be seen from these results, the response was generally positive, with most of the audience enjoying the experience and expressing an interest in availing themselves of such a system if it were available. Our criteria for the game, simple, casual and engaging were shown to have been accomplished to a greater or lesser extent. In excess of half of the audience felt the interaction was intuitive and that the rules were easily understandable from playing the game; supporting our claim that the game is simple to understand and join in. A considerable majority answered that the game was entertaining and that they would play again given the opportunity, solidly supporting our claim that the game is engaging.

We also observed, much as indicated by Maynes-Aminzade, Pausch and Seitz [2, Section 5.3] that the majority of the crowd was engaged in the activity, even when not directly interacting with the ball. Seemingly, an implicit cooperation was quickly realized; parts of the crowd on the

Question	raised hands as a percentage
Was this entertaining?	82
Would you use such a system if it was available in some public location?	73
Were the rules easily understandable from the explanation?	70
Were the rules easily understandable from playing the game?	56
Did your interaction with the game feel intuitive?	58
How many feel it was simple to get the ball to move as you wished?	28
How many feel it was too hard?	25
How many of you spent too much time unable to affect the ball?	34
How many preferred the regular video representation?	86
How many preferred the difference image?	8

Table 1: HGC audience response

same side of the line support each other in a common goal, passing the ball around. There was the occasional shouted comment, but these seemed to be between audience members already familiar with each other. However, there was also a lower-order communication: the crowd producing a combined murmur of encouragement or excitement when the game went in the favor of one team or during periods when control of the ball was hotly contested.

The criterion that the came should be casual was not as well tested during the presentation, given that the audience was told they were about to play a game, and as such, most members engaged in the activity. However, during a later lull, while we were all waiting for a decision from the jury, we set up the game to simply run without any announcments. During this period, members of the audience were walking freely about, some leaving the room to get food or drink, some remaining seated, some gatherin in small groups. Even in this state, those who were interested were able to play the game, while others ignored it, some switching between playing and other activities, some entering the room and joining in spontaneously. While less documented, we hold that this is a good indication that we achieved an appropriate level of casual involvement.

Another important claim can be made about the results: from the responses to the questionnaire, as well as observing the crowd while playing, the audience reacted similarily to what was indicated in *BallBouncer* [3, Section 4], which is the primary feature in the evaluation of this project; at least indicating, if not proving, that our framework can be used to achieve the same sort of results.

Regarding our secondary intent, this test is a good indication that game prototyping can be useful in acquring experimental data. The audience was not only enthusiastic about engaging in the activity provided, but seemed positive to the idea of answering a questionnaire concerning their experience. In this instance the responses to the questions about the video representation are of special interest in that they felt the visual feedback was important in interacting with the game. Several people commented that it was too hard to see who was where in the difference image. This is an excellent example of feedback on an interaction feature, and underlined by the nearly unanimous agreement.

The more informal tests with smaller groups produced much the same results, though these tests are qualitatively different not only in the size of the subject groups and the lack of a formal

questionnaire, but also in the fact that the smaller test groups had been recruited for the sole purpose of testing an AR application. Nonetheless, the reactions gleaned from these tests, and indeed the completely informal tests run on visiting family members bear out the conclusion that even quite simple prototypes will encourage an audience to provide interaction data given that it is engaging. Also, and more importantly for our project, that our framework is capable of producing these kinds of prototypes.

4.1.2 Choices

Unlike most other augmented reality projects here referenced, VARG A.R.E.N.A. does not require any equipment beyond the computer and one or more cameras. This is because the game is designed to be played by a large group of people, and the distribution of such items to each player would prove costly, inefficient, and unnecessary. Supposing the items had any kind of value, there would also be the problem of theft. Instead, our system uses merely the movement of the crowd to interface with the system. Certainly, such items may be an idea for a future extension of the system, but at the moment that is not implemented nor planned. It is also worth noting that some artifacts may prove useful, in that the system detects movement by comparing two subsequent frames on a pixel-by-pixel basis. Examples would be to use a white jacket if the seats or surrounding players are dark of color.

Though some other augmented reality projects also are designed for stationary operation, this is not usually the case. Limiting ourselves to such a setup has great gains in flexibility in other areas, one of which is that the detection of movement is local only, and no cycles go to waste determining the orientation of the system. In addition, this allows us to create a more detailed and correct model of the environment, in that we have a priori knowledge of its design, and it is unlikely to change suddenly and gratuitously.

Our system assigns no value to individuals - only movement, which may be conducted by one or more persons. This allows us to give an immediate feeling of team spirit, by dividing the audience into visually distinct groups. The game is then played with the goal of furthering the goals of the group as a whole. Arguably, one could set out to sabotage this by working against the implicitly given rules, but the lack of focus on individuals would render such an effort largely futile.

While the application is designed chiefly for inclined surfaces with a seated audience, this is not a necessity for its operation. The system is flexible enough to accommodate surfaces of any topology with some adjustment. This possibility is not implemented in the current application, but its addition would be trivial. In addition, movement need not necessarily be restricted in the way stadium seating would. One could certainly imagine the system utilized with a flat, planar surface as the play field. It is worth pointing out that a crowd running around while keeping their eyes on a screen could increase the potential for physical harm as absent-minded individuals focus solely on the image rather than their own movement relative to the surrounding mob. Particulary this could prove dangerous in the case of the audience being teenagers or little children, as they are notorious for their recklessness. Some supervision might be advisable should this be the case. It is also it is important that it be devoid of clutter, that the audience not lose their footing upon hitting unexpected protrusions or sudden depressions.

As for locations, our application is designed for indoor usage. By imposing this restriction, we define away several complex issues in favor of a much more controllable environment [6, Section 2.2]. Ambient light conditions greatly effect the image processing; in particular, any sudden change in brightness is interpreted as movement, and thus the passage of a cloud would to the system be equivalent to the whole audience jumping. Inside a closed hall, this is rarely a problem. Other issues with outdoor usage is the effects of weather. As video is our only input, anything disturbing the feed will confuse the system. Rain and other precipitation might be

thought of as having the effect of random, potentially massive noise.

4.2 Shortcomings

Out of three milestones, we only made one. The first failure was due to a necessary reevaluation of our project plans: we originally intended to replace sections of the original program piecemeal, when this didn't work out, we had to start building the program from scratch, which made it more or less impossible to have a working copy ready for the first milestone. The second milestone we should have made, and we were close, but the persistent issues regarding the video stream delayed progress too much. We consider the issues surrounding the video stream, and the massive delay resulting from them, to be our greatest failure, and to no little extent, it is responsible for our failure to extend the basic functionality.

While the extended functionality was not part of the central goal set out initially, our failure to move beyond the basic framework was a disappointment. As it is, while the code is certainly more maintainable and readable than the original, the current state of the project can only produce a somewhat stunted version of the original efforts in *BallBouncer*. As such, the program will need some more work before it satsifies the needs of our sponsor. One of the major planned features that was omitted by necessity was the improved movement detection; as is outlined in Section 3.4, we had to abandon the attempt to run the processing on GPU, and consequently, performance sank to unacceptable levels given our criteria of 30 frames per second at 800 by 600, as stated in Section 3.2

Finally, we were less strict about the work routines during the middle period of the project, resulting in poor progress. This was in part an effect of the mounting frustration with the video issues, but also a cause for prolonging them. A strong supervisor would have been invaluable during this period, but unfortunately, at that point, due to reasons mentioned in Section 2.1 we were in between supervisors.

Several things did not go as planned in our execution of the project, most glaring of which was the lack of discipline that evolved, perhaps due to the decision to work from home rather than at school. We had reasons for this decision, as most good workplaces there were already taken, and the less good ones were, well, less good.

In the beginning, working at home was at least as good as either of the choices available to us at school, and discipline was good. However, lacking any external pressure to work, discipline slowly degenerated. Coupled with the video stream's persistent stubbornness in refusing to work, this led to somewhat lax working habit, at least in periods.

This problem would have been ameliorated by stricter routines, a procedure that would have been promoted by having to leave the safety of our home, or by having someone watch over us, to whom we would feel obliged to listen. It is our belief that our supervisor was supposed to take on this mantle, but misunderstandings and a general lack of knowledge of what was expected lead to this not happening. When later we changed supervisor, the notion that the supervisor did not supervise, was ingrained and not questioned, and perhaps more importantly, work had become very hectic to meet the upcoming milestone.

4.3 Further development

As regards further developent, there are extensive possibilites, both technical and academic in aspect. Regarding the technical aspects there are two main areas of improvement available: im-

proving the modules already present, or extending functionality by adding new modules. While we do hold pride in our work, we must admit there are some areas in the code which we would have liked to put some more work into. Chief among the areas lacking in polish would be those added just prior to the presentation at HGC, as the rush did somewhat inhibit good coding practices, leaving particularily the function handling the force calculation bloated with responsibilities that should be handled elsewhere in the code to avoid coupling the modules too tightly. However, one of the main priorities in creating the framework was that it should be a simple matter to add later extensions. The modular nature of the code supports this concept well, and though it would benefit from a period of proper cleanup and polish, extending the basic framework through additional classes is quite possible, and indeed provided for.

One of the most important intended features that we had to omit due to time constraints was the improved movement detection. The movement detection in *BallBouncer* [3, section 3.1] is relatively simplistic; our approach is much the same, and is described in Section 3.4, on implementation. The current detection has no facility for truly detecting the direction of movement, instead relying on what amounts to guesswork. While this is acceptable in many cases, the occasional mistake is noticeable, especially with large, quick movements, where the area of motion passes through the area of the ball between one frame and the next, and the program, knowing only that there is now motion on the other side of the ball ends up interpreting it as motion in the opposite direction. Our solution, which never progressed beyond concept, was to analyze the current as well as the previous difference image created, so that the program could compare two sequential difference images and identify in which direction motion areas had moved. The specific technical aspects of this operation were never clearly defined, but any attempt to improve the basic framework would be well served by working out the process and implementing it, as this would provide more convincing interactions between real and virtual objects; a central ideal for this program. In order to not abandon the principle that the program run smoothly on an average laptop however, our issues in moving the processing from CPU to GPU, outlined in Sections 3.4 and 4.2 would have to be overcome.

Leaving aside the technical aspects though, the main purpose of the framework is to facilitate simple, rapid prototyping of AR games. As such, it would be a natural base for projects that sought to investigate alternate interfaces, crowd computer interaction, certain social aspects of IT or a multitude of other topics. Though some measure of game theory would undoubtedly be invaluable, an element of anthropology, sociology or other human science could give a much firmer foundation for analyzing the behaviour of the test audiences and drawing conclusions about e.g. the effects of AR interfaces.

As ARToolKit has the ability to recognize special shapes and determine their distance and orientation, a planned feature was to utilize this capability to automatically determine the extent of the locale in which the system was set up, as well as the orientation of the camera used. With the abandonment of ARToolKit for video display purposes, and the time spent getting FFmpeg to do that task, this was pushed down the priority list. If it were to be added, ARToolKit is perfectly capable of processing still images from other sources, like FFmpeg. We feel it would be a very interesting extension to be added in the future. However, given the stationary nature of the application, such an addition may prove to be an unnecessary luxury, as inputting these values

would be a one-time job in all but the rarest of cases.

4.4 Group evaluation

4.4.1 Organization

Considering the organization of the project, we are reasonably pleased; though our team was small and our acquaintance made for an informal atmosphere, we mostly adhered to good principles of software development. The main deficiency, as has been intimated earlier, was the lack of a dedicated organizer; a team leader who could dedicate their time to distributing responsibilities and keeping tabs on progress.

4.4.2 Distribution of work

During the development, we were each responsible for certain classes. Though it must be admitted that we each interfered some in the other's work, this was largely beneficial, due to good communication, and we kept to the responsibilities, so that should one of us require a certain feature in a class the other was responsible for, the common approach was to inform the other member and await resolution. As Simen is a significantly more experienced programmer; he to some extent took on the role of lead programmer, having "the majority vote" when we decided matters of design and implementation. Arild, on occasion provided a result-oriented perspective when

4.4.3 Subjective experience of the Bachelor thesis

Arild: In hindsight, we might well have been too focused on the development of the program to the point that we lost sight of the project beyond. During the latter period of the project, when we turned our focus to research and the writing of this paper, I found that there was a lot of interesting subjects I would have liked to relate to this project. Primarily the field of human-machine interaction, which is seeing interesting experimentation these days. In some ways, it is a point of pride that we completed our task while overcoming difficulties, in another way, I regret the time we could have spent extending this project further. Overall, though it has been both enjoyable and I feel I have grown as both a programmer and a software development.

Simen: Working on a team has certainly been a learning experience, and the task was interesting and challenging. There were times when our problems overshadowed the joy of programming, but those moments were rare. As Arild mentions, we are programmers first and foremost, and the programming was to us perhaps too absorbing. All in all, the project has been enjoyable, and I would like to continue the task in the future.

5 Conclusion

As an experiment in either the use of games as a testbed for experiments, or the use of AR as an interface, this project would be sadly lacking. However, these are secondary and tertiary objectives to the main purpose of this project. As regards the main purpose, we have produced an extensive framework, which has proven itself capable of performing its intended function; the creation of AR-based games. This proof takes the form of the creation of a demonstration game which was implemented over a short period of time, and was subsequently tested and found to be acceptable in engaging an audience.

Furthermore, we have to some extent substantiated the claim that rapid prototyping of games can be a viable method for acquiring experimental data; our demonstration game was prototyped over a brief period, and was able to engage an audience to the point where we could acquire relevant data.

During the project we acquired a great deal of experience, largely as regards programming. One of the central lessons was that when working on a complex system, it is doubly important for any code produced to adhere to good coding standards, as others rely on the sections you are responsible for, and overall team efficiency is limited if they can not easily grasp the modules they need. Related to this is the desire to alter modules which other members are responsible for, even with the aim of improving their standard, this can easily have a cascading effect on the work done by others, delaying the entire project considerably.

In addition, we learned several lessons on the subject of development methodologies and software development in general, and specifically about Agile methodologies, as exemplified by Scrum. The fluid nature of the design and implementation suited the project well in general, but that same nature makes it difficult to correctly estimate the course of development, which can lead do significant difficulties in meeting fixed milestones. Related to this is the importance of the critical path; though our project is highly modular, the delay of the central video feed module left us unable to proceed with testing, which effectively halted further development for an extended period.

5.1 Final summary

In conclusion then, in all our tests, small or large, players took to the experience intuitively, experimentally and spontaneously. As well as indicating that our framework fullfills the basic requirements made of it, these tests substantiated the viability of using games created this way as a testing ground. And furthermore, apart from any other consideration, the enthusiasm shown for these kinds of games by any audience is still inspiring, and more than anything is a motivation for further work in this field.

Bibliography

[1] M. Billinghurst, I. Poupyrev, K. H. & May, R. 2000. Mixing realities in shared space: An augmented reality interface for collaborative computing. In *ICME*, 1641–1644.

[2] D. Maynes-aminzade, R. P. & Seitz, S. 2002. Techniques for interactive audience participation. http://www.monzy.org/audience/ICMI-2002-finalpub.pdf.

[3] J. Sieber, S. M. & Wyvill, G. 2009. Ballbouncer: Interactive games for theater audiences. http://metamanda.com/crowdcomputing/subs/sieber-mccallum-wyvill.pdf. (Visited May. 2010).

[4] T. Starner, B. Leibe, B. S. & Pair, J. 2000. Mind-warping: Towards creating a compelling collaborative augmented reality game. In *Proceedings Intelligent User Interfaces (IUI) 2000*, 256–259. ACM Press.

[5] Jegers, K. 2007. Pervasive gameflow: Understanding player enjoyment in pervasive gaming. http://www.ipsi.fraunhofer.de/ambiente/pergames2006/final/PG_Jegers_GameFlow.pdf.

[6] B. Avery, W. Piekarski, J. W. & Thomas, B. H. 2006. Evaluation of user satisfaction and learnability for outdoor augmented reality gaming. http://www.tinmith.net/papers/avery-auic-2006.pdf. (Visited May. 2010).