Ducking and Diving

Alexander Zierbeck
August 2010

MSc Artificial Intelligence
Supervisor: Ruth Aylett
Declaration

I, Alexander Zierbeck, confirm that this work submitted for assessment is my own and is expressed in my own words. Any uses made within it of the words of other authors in any form e.g., ideas, equations, figures, text, tables, programs etc are properly acknowledged. A list of references employed is included.

Edinburgh, August 19, 2010

Signed: .........................
Abstract

The project “Ducking and Diving” is basically a real time motion capturing project which takes part in the sector of Virtual Environment (VE). It combines human-computer interaction with virtual reality.

The task of this project is to downsize the gab between the real and the virtual world.

This project is realized as a game where player is walking in front of a screen and tries to avoid virtual obstacles which are moving towards him. The player should see the virtual environment as an extension to the real world, which behave correctly while moving around.

To increase the immersion, the project is using a projection system, which enables the player to see the virtual world in 3 dimensions (3D).

Furthermore a motion capturing system is used to capture the movement of the player and transfer that motion onto the virtual representation of the player on the screen. Now this avatar can be steered by the own movement.

The main goal of this project is to evaluate, if interaction with a virtual environment in real time is possible and applicable for such an application which combines video games with sports.
Acknowledgements

This thesis with the title “Ducking and Diving” is the final project of my study to achieve the degree of a Master of Science in Artificial Intelligence. The project was created in the period from the 24th of May 2010 till 19th of August 2010 at the Heriot Watt University, Edinburgh, Scotland.

In this section, I would like to thank all the people who supported me with my project.

First of all, I would like to thank Ruth Aylett, my supervisor, for pushing me in the right direction with her advises and suggestions.

Also I would like to thank my colleagues Ainara Azcona and Markus Nehfischer who helped me with important advices and ideas. This was very significant for me and for the project.

At last, a big thanks to my family and friends, who supported me at any time within my masters year and a special thanks to my parents, who made it possible for me to study this year here in Scotland.
Contents

<table>
<thead>
<tr>
<th>Content</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Motivation and project definitions</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Formal hints for this document</td>
<td>3</td>
</tr>
<tr>
<td>1.3 Structure of this document</td>
<td>4</td>
</tr>
<tr>
<td>2 Literature review</td>
<td>5</td>
</tr>
<tr>
<td>2.1 Overview</td>
<td>5</td>
</tr>
<tr>
<td>2.2 Background research</td>
<td>6</td>
</tr>
<tr>
<td>2.2.1 Creating the virtual environment</td>
<td>6</td>
</tr>
<tr>
<td>2.2.2 Collision detection</td>
<td>8</td>
</tr>
<tr>
<td>2.2.3 Motion capturing</td>
<td>9</td>
</tr>
<tr>
<td>2.2.4 Head tracking</td>
<td>11</td>
</tr>
<tr>
<td>2.2.5 3D Projectors</td>
<td>13</td>
</tr>
<tr>
<td>2.3 Literature summary and assessment</td>
<td>14</td>
</tr>
<tr>
<td>2.4 Conclusion and discussion</td>
<td>14</td>
</tr>
<tr>
<td>3 Conception and design</td>
<td>16</td>
</tr>
<tr>
<td>3.1 Requirements</td>
<td>16</td>
</tr>
<tr>
<td>3.2 The tracking system</td>
<td>17</td>
</tr>
<tr>
<td>3.2.1 The tracking software</td>
<td>18</td>
</tr>
<tr>
<td>3.2.2 Placement of the markers</td>
<td>19</td>
</tr>
<tr>
<td>3.2.3 Placement of the cameras</td>
<td>20</td>
</tr>
<tr>
<td>3.3 The projection system</td>
<td>21</td>
</tr>
<tr>
<td>3.4 The control software</td>
<td>21</td>
</tr>
<tr>
<td>3.4.1 Integration of the motion data</td>
<td>22</td>
</tr>
<tr>
<td>3.4.2 OpenSceneGraph</td>
<td>23</td>
</tr>
<tr>
<td>4 Implementation</td>
<td>25</td>
</tr>
<tr>
<td>4.1 Introduction</td>
<td>25</td>
</tr>
<tr>
<td>4.2 The main function</td>
<td>26</td>
</tr>
</tbody>
</table>
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.3</td>
<td>Implementation of the tracking client</td>
<td>30</td>
</tr>
<tr>
<td>4.3.1</td>
<td>The RTClient</td>
<td>30</td>
</tr>
<tr>
<td>4.3.2</td>
<td>Additional components</td>
<td>32</td>
</tr>
<tr>
<td>4.4</td>
<td>Implementation of OpenSceneGraph</td>
<td>34</td>
</tr>
<tr>
<td>4.4.1</td>
<td>The basic classes</td>
<td>34</td>
</tr>
<tr>
<td>4.4.2</td>
<td>Helper classes</td>
<td>40</td>
</tr>
<tr>
<td>4.4.3</td>
<td>Scene graph chart</td>
<td>43</td>
</tr>
<tr>
<td>4.5</td>
<td>The obstacles</td>
<td>44</td>
</tr>
<tr>
<td>5</td>
<td>Results and evaluation</td>
<td>46</td>
</tr>
<tr>
<td>5.1</td>
<td>Introduction</td>
<td>46</td>
</tr>
<tr>
<td>5.2</td>
<td>Usage and instructions</td>
<td>47</td>
</tr>
<tr>
<td>5.3</td>
<td>The first prototype</td>
<td>49</td>
</tr>
<tr>
<td>5.3.1</td>
<td>Screen shots</td>
<td>49</td>
</tr>
<tr>
<td>5.3.2</td>
<td>Evaluation</td>
<td>50</td>
</tr>
<tr>
<td>5.3.3</td>
<td>Summary</td>
<td>53</td>
</tr>
<tr>
<td>5.4</td>
<td>The second prototype</td>
<td>54</td>
</tr>
<tr>
<td>5.4.1</td>
<td>Screen shots</td>
<td>55</td>
</tr>
<tr>
<td>5.4.2</td>
<td>Evaluation</td>
<td>56</td>
</tr>
<tr>
<td>5.4.3</td>
<td>Summary</td>
<td>59</td>
</tr>
<tr>
<td>5.5</td>
<td>Evaluation results</td>
<td>60</td>
</tr>
<tr>
<td>6</td>
<td>Final conclusion and future work</td>
<td>62</td>
</tr>
<tr>
<td>6.1</td>
<td>Summary</td>
<td>62</td>
</tr>
<tr>
<td>6.2</td>
<td>Future developments and ideas</td>
<td>64</td>
</tr>
<tr>
<td>6.3</td>
<td>Personal conclusion</td>
<td>66</td>
</tr>
</tbody>
</table>

Bibliography | III |
List of Tables | V |
List of Figures | VI |
Code Listings | VIII |
Acronyms | IX |
Appendix | X |
A.1 | Feedback sheets | X |
A.1.1 | Questionnaire prototype I | XI |
A.1.2 | Questionnaire prototype II | XIII |
The first chapter of this thesis will introduce the project and will give an overview of this document. The first section will explain the project requirements and why this research project has been launched. After that, there will be some instructions and formal hits for this document. At the end, the chapters of this document will be introduced.

1.1 Motivation and project definitions

With help of fast computers we can create amazing VEs, that look almost like the real world. The gap between virtuality and reality seems to get smaller and smaller. But this basically affects the visual part and most times it is restricted to a monitor or a canvas in two dimensions.

![Figure 1.1: Screenshot from the project "Virtual Tübingen" [webh]](image)

In the last few years, new technologies are developed to enhance the visual impression. An important development is the usage of 3D monitors and projectors. Now you can see the VE almost like you see the environment in reality. This helps very much in terms of immersion.
A more advanced system is called the “cave” where people can walk in a small room and the environment is projected on all four walls.

But what about the interaction with a VE? In a cave you can walk around and see the VE almost like the real world but grabbing a virtual object for example is still pretty difficult. There are a lot of tools to get input from the user and to transmit it to the VE (for example a data glove). Also feedback towards the user is important.

One important and technically mature method, to get the movement of a person and capture the data, are motion capturing systems. They can record every motion (depending on the amount of markers and placement) of a person and process the data. This is used very commonly in the film industry for special effects. But this is completely without interaction from the user because films are a single way medium.

Can such a system be used to combine it with a VE, so that the user can walk around in a virtual world? And what about the virtual objects in the scene? Can the user interact with them?

This research project is going to find out, if a user can walk in a VE in real time and if interaction with virtual objects can be applied. This interaction only applies for collision detection.

The project is planned as a game. The user can walk around in a predefined area and can see the world in 3D. The interaction with objects is realized in that way, that the user has to avoid obstacles which are coming towards him.
1. INTRODUCTION

Another fact for this implementation is, that the video gaming industry is going more and more in the direction of active user interaction and interactive sport where the user has to move himself to control the game.

The main goal of this project is to implement a functionality, where the movement of a person is tracked and directly displayed in a VE. Also a collision detection has to be implemented to indicate the intersection of player and obstacle. All of this should be combined with a visual component.

1.2 Formal hints for this document

This document is written in British English and uses many technical words and phrases which are explained where they occur or in the glossary.

Important parts of the code will be shown and explained in this document. The code fragments will marked as such and will be listed in the code listing section at the end of this document.

This document is created with the document preparation system \LaTeX (MiKTeX version 2.8). As graphical user interface and editor, TeXnicCenter (version 1.0) was used.

The graphical illustrations are created with the program Inkscape (version 0.47) and Paint.Net (version 3.5.1). Photos are edited with Paint.Net (version 3.5.1) and Gimp (version 2.6.10). Class diagrams are made directly in Mircosoft Visual Studio (version 9 / 2008) and are adjusted with the mentioned graphic editors where necessary.

To emphasise important content in this document, the following text styles are used:

- **bold**: content, with high relevance is displayed bold
- **italic**: further important terms and phrases are shown italic
- **typewriter**: for source code, the text format “typewrite” is used

The terms OSG and OpenSceneGraph are used pretty often in this document. They are both meaning the same. OSG is just a short form for OpenSceneGraph. So there is no difference between these two terms.
1.3 Structure of this document

The document is basically divided into four main parts: the literature review, a theoretical / conception part, the implementation and the evaluation / conclusion.

The literature review describes, what was done in the past by other people according to this project and which parts of this project are most important. The conception is taking the information of the literature review to plan the steps for the implementation. The implementation itself describes, how the program was realized and how the modules are combined and are working together. The last main part of this document is the evaluation and the review about the project. The conclusion will summarize and end the project.

Chapter 1 – Introduction: The introduction will explain, what this project is about and will describe the motivation and requirements. Also explained are formal hints and tools. The end of chapter one presents the structure of this document.

Chapter 2 – Literature review: The literature review reveals, what relevant projects has been done before and what methods and tools they used. It is a collection of information, which is important to develop this project. This chapter closes with a summary and a short plan of discovered main topics for the project. Also first requirements are defined.

Chapter 3 – Conception and design: The conception takes the information gathered in the literature review and plans the modules, that are going to be programmed. Also some information about the hardware, that is going to be used, is provided. Some considerations about the placement of the markers and the cameras are done as well.

Chapter 4 – Implementation: This chapter will explain how the project is programmed according to the conception in the chapter before. Class diagrams will be displayed and explained to show the structure. At the end, a scene graph will show the elements of the scene and the arrangement.

Chapter 5 – Results and evaluation: The evaluation will reveal, how good the program is. This is done by letting a group of people test the program and analyse their feedback. Two prototypes are planned to implement some ideas from the participants.

Chapter 6 – Final conclusion and future work: The conclusion summarizes the project and will explain the results. Also some future development opportunities will be described. The personal conclusion will end this document.
CHAPTER 2

Literature review

The second chapter of this document is the literature review. This chapter is going to evaluate some related projects and how they could help in the own process of designing and implementing. It will also reveal the most important elements of this project and how they could be realized.

2.1 Overview

To start the literature review, the project has to be divided into its main components. After that, the individual parts can be investigated and planned. It is also more likely to find more information on small parts of this project instead of searching for similar experiments. The introduction of this document explained the project definitions. Also the discussion with the supervisor and the existing equipment in the lab led to the following main topics:

- Programming a control program in C++
- The graphical component
- How to implement the collision detection
- Real time motion capturing with infra red (IR) cameras
- Head tracking
- 3D projectors
Figure 2.1 is illustrating some of the main components. This is just a simple diagram to show, how they work together. Later in the project, a more sophisticated design will be presented, after testing the hardware.

2.2 Background research

This section will investigate the main topics of this project. Basically it is about searching for information and other similar projects, or parts of other projects. In this case, not every component has to be developed from the scratch and existing ideas can be used and extended for the own project. Acknowledgements are made in the text.

2.2.1 Creating the virtual environment

The VE will be programmed in C++ and OpenSceneGraph [webg]. The scene graph system will be used, because it is relatively easy to create a simple environment. Also in combination with C++ it is a fast working tool which is essential for this project. OpenSceneGraph is using OpenGL, what means, that the visuals are programmed very close to the hardware. It is also not bound to Windows computer but can also be used on systems like Linux or OSX. It is also well established as “the world leading scene graph technology” [webg] which means, that the community and the help is pretty distinctive.

But why not using a game engine which is also pretty fast? Game engines, or the graphic part of it, are in most of the cases pretty complex. They consist of parts which are essential for games but not really for a virtual environment. For instance: sound system, steering control, data management (saving / loading a game). These things can be useful in a virtual environment but are
Not needed in terms of this project. Furthermore, game engines are not very easy to handle. The programming is often very complex and takes some time to get into it. Also the hardware requirements are higher because of all the parts, that are needed to build a game. The good thing about game engines is, that they are developed to create games. That means that they have everything needed to build an virtual environment and in most of the cases they have a physics system. They are also pretty fast because they are designed for the graphics interfaces OpenGL / DirectX. That leads to a good looking environment, programmed in C++.

So it’s not really sure which engine to use. Preferable is OpenSceneGraph because of the simplicity. But it’s worth, to give a game engine like Ogre [webf], Irrlicht [webc], or the Quake engine [webj] a try.

A good introduction into game engines is the book “3D game engine design: a practical approach to real-time computer graphics” written by David H. Eberly [Ebe07]. It shows pretty good basics to develop games and VEs.

For the further research it will be assumed to work with the scene graph system OpenSceneGraph. A scene graph is a comfortable way to organise the objects in the scene. “It is a directed acyclic graph – a hierarchical relationship among the nodes” [Bar05]. OpenSceneGraph provides everything what is needed to create a VE. All basic shapes can be inserted, even complex models can be loaded. It is also (of course) possible to translate, rotate and scale objects. For this project it is enough to work with the simple shapes. The focus is more on the usability and collision, not on the visuals at first. It is also better to apply the calculation power on other things like processing the movement and collision detection.

Figure 2.2 is showing an example, how the environment could look like. It’s kept very simple. The arrows indicate, that the objects (the grey blocks) are coming towards you and you have to avoid colliding with them by walking around or jumping over.

Concluding, the main objects that are used are at first is a plane as the floor and boxes as the obstacles. Later on, a menu will be integrated to control the game. This will also be visualised in OpenSceneGraph and controlled by a mouse or a other device. For that, OpenSceneGraph also provides Text-elements [webg] and event handlers.
Furthermore the representation of the player has to be considered. The first idea would be very simple. For example, a third person view and the user is represented as a box or a cylinder where the feet are bottom edge and the head the top edge. In further developments this representation can be advanced to a more human like level.

### 2.2.2 Collision detection

A big part will be the collision detection. OpenSceneGraph already provides a mechanism which has to be investigated [webg]. It is using bounding boxes and intersection tests. If this does not work properly, an own technique has to be implemented.

A simple and exact algorithm has been developed by K. Chung and W. Wang [KC96] or a low bandwidth algorithm called CULLIDE [NKGM03]. CULLIDE uses two levels to detect a collision. First thing to check is, if there is a potential collision. This is done by testing if the bounding boxes of the objects intersect. The bounding box is a non visible box, where the object fits in exactly. If two bounding boxes collide the system will determine an exact collision. It will localise, where the objects hit each other exactly. See Figure 2.3 for illustration.

This will save calculation power because not every object is fully tested. A bounding box collision check is very cheap but not very accurate.

![Figure 2.3: Different stages of a collision](image)

What collision detection algorithm is used will be determined by, how exact the collision should be and of course the power of the computer. The collision detection is very extensive work. In case of this project, the collision detection is an essential part and has to work really properly and exact, in order to give the user a correct feeling for this application.

Here also the book “3D game engine design: a practical approach to real-time computer graphics” written by David H. Eberly [Ebe07] is giving some advices for collision detection (chapter 8). He describes topics like “finding collision between moving objects” or “a dynamic collision detection system”. This could also be very interesting for the implementation, if the other systems don’t work properly.

First of all, the implemented algorithm in OpenSceneGraph is going to be tested.
2.2.3 Motion capturing

This project will be equipped with an IR motion capturing system. That means, it is an *optical* system which is working with light reflections. The infra red light is reflected by special patches, also called marker [Nus06]. Infra red light is used, to not disturb the user while acting because infra red light is invisible for the human eye.

To determine the position of the markers, you need more than one camera to ‘see’ the marker from different angles in the 3D space [Nus06]. The problem is, that markers are not always visible when the user is moving. It is important to place the cameras in such a way, that the markers are visible to the cameras at any time, no matter how the user is moving. Also the number of cameras should not be less than four and no more than 32 to get a proper working system [Men00].

It is also possible to track more than one subjects at the same time. That makes it possible for two persons to use the system simultaneously. That can be established because the infra red motion capturing system can deal with a large number of markers (up to 200) [KW08]. In case of this project, a further development could be a multi player mode.

Also the cameras have to be calibrated every time they got moved. This has to be done because the cameras triangulate the position and to get correct and exact results. For that, there are special measurement shapes with markers which have to be moved in front of the cameras for a certain time.

IR cameras have several pros and cons [Nus06] [Men00]:

**Advantages**

- precise
- absolute position determination
- can deal with many markers
- no attached cables to the performer
- high sample rate

**Disadvantages**

- expensive (the cameras)
- always on the same location
- limited reach
- can loose the tracking to markers
- noise from other light sources (especially sun light) can have bad influence
This technique of motion capturing would be pretty useful because the player doesn’t have cables or bulky equipment attached. Perfect would be some kind of video motion capturing. In that case, the player also doesn’t have to wear markers.

To make the different steps of motion capturing and processing more descriptive, the following picture will depict four important stages:

![Figure 2.4: How motion capturing works](image)

The picture (Figure 2.4) is showing the process of capturing the movement of a real person and putting it on a virtual character. On the right side is a user with attached markers (the red bubbles). The next step shows only the markers, the motion that has been captured. This motion is then rigged to a simple and even more complex virtual character.

There are many different motion capturing methods (optical, magnetic, mechanical, hybrid) but the reason why the optical system (IR cameras) is used for this project is simply because they are available in the lab and designated for this project.

**Real time motion capturing**

Basically, when you are working with motion capturing systems (which are actually not designed for real time applications), you need a lot of post processing to make the visual character look more real. This is something that can’t be done (or only with a lot of effort) in real time. But real time motion capturing is essential when you want to move a virtual character in a VE in real time.

In order to realise that, you either need a very fast computer or you don’t set so much value...
on how the movement looks like. For the second part it means, that you leave out the post processing and get the movement-data from the cameras directly into the virtual environment with less correction and adjustments.

A project [TH98] is doing tracking and rendering in real time. They use a couple of cameras to capture the movement via pattern recognition. They are also using a Kalman filter to reduce noise, produced by the tracking. This can also be important for this project when using the IR cameras to get more exact results for the calculation for the collision detection. A similar project [DU03] is trying to enhance the tracked movement by using Kalman filter and to compensate lags with the tracking. With that, they are creating a exact and robust tracking system.

Important to say is, that this project don’t set much value on how the movement and the models look on the screen. More crucial is, that the results are accurate and exact. The tracking software is also already smoothing the data.

### 2.2.4 Head tracking

Head tracking is a possibility, to track the movement of the head in 3D space. You could track up to 6 Degree of freedom (DOF), which are the three angles and the position in 3D space. With this motion data you can adjust the camera in a VE in order to let the user look around objects, just like he can do in the real world. This would also give some kind of 3D effect (3D though movement).

For this project it would be very helpful because the user can look at the obstacles, which are coming towards him, from different angles. This makes it easier to estimate the distances and the speed so that he can successfully avoid the obstacles.

There are many different methods for head tracking. Most common is the tracking with an optical system (see the previous section 2.2.3). The user has to wear a special marker pattern on the head.

The fact, that the project is using a passive stereo system and the user has to wear special glasses, offers the possibility to attach the markers to the glasses like shown in figure 2.5.

One project called “Real Time Tracking of High Speed Movements in the Context of a Table Tennis Application” [SR05] invented this type of glasses. The markers they are using are pretty big but this is necessary for fast movements.

The marker pattern also enables the system to use 6 DOF. So yaw, pitch and roll can also be calculated. But the 6DOF are not required for this project. Only the position and the movement are important.
Other projects are using just a webcam to track the movement of the head. The software recognises the face and calculates the motion data with checking, how the pattern of the face is changing. This is a pretty cheap solution because you only need a webcam and a proper software or an Application Programming Interface (API).

One API is the FaceAPI [webb]. It is able to process 6DOF data in real time and can also recognise facial expressions. It is also non commercial and free to use.

The problem with this technique to use it in this project is probably, that the user is too far away from the webcam. So it can not recognise the user any more.

Another, more unconventional solution for head tracking, is the idea from Johnny Lee [webd]. He is using a Wii remote (an optical system), but the other way around. So the Wii remote is laying near the screen and is receiving the signals (infra red light) from the sensor bar which is mounted to the head of the user. With this combination, the position of the head can be determined. A control program (free for download from Johnny Lee’s homepage) is doing the calculations. This is a very interesting solution but not applicable for this project because the tracking system is also using infra red light and that would disturb the Wii remote.

Considering the problems of the three methods, the first one with the optical system and the marker pattern would be the best one because this system will be used anyway. The head tracking can be implemented into this system without problems.
2.2.5 3D Projectors

This project is using a virtual environment which will be displayed in 3D. For that, a passive stereo system will be used. That means, that two projectors and polarising filters in front of the lenses are used. You also need a pair of very simple and cheap glasses.

One projector will display an image for one eye, the other one for the other eye. The light will be polarised by the filters in front of the lenses and the glasses will only let the associated light through. In other words, each eye is receiving a different image. This will add depth to the scene and create a 3D image. The passive stereo system is cheaper than the active stereo. For the active stereo system you need special shutter glasses which are pretty expensive. Also the passive stereo system can handle higher frequencies which avoids flickering [DP].

![Passive Stereo System Chart]

Figure 2.7: The passive stereo system chart

Figure 2.7 is showing the data flow of the passive stereo system described before [Her04].

Also to mention is that the system is a front projection. So the canvas, that is being used, is not a special material. The computer in the image needs a graphic card with two channels (one for each projector) or two graphic cards. This is meant by the splitter component in the figure 2.7 to get the two different images. The control of the projectors is done by the driver directly on the host system. So you can switch for example between 3D and non-3D projection.

The alignment of the projectors can vary, depending on the usage [DP]. The situation in the lab is that the two projectors are mounted on the ceiling. They are pointing in the same direction but
slightly shifted and they have to have the same resolution. They need to be adjusted very well, otherwise you won’t get a correct 3D effect.

2.3 Literature summary and assessment

The literature review has revealed the important parts of this project and the sources, that can help in the process of implementing the system. The components, that have been described should not be too hard to assemble. More complicated is putting all this components together and let them work as one unit.

Also problematic can be the collision detection part. As described, OpenSceneGraph already has a mechanism implemented but it’s not sure, if this works properly. Otherwise other algorithms have to be used. The fact, that this is a real time application, the collision detection has to work fast and precise.

Another problem, that maybe will come up, is the latency time of the motion capturing. The processing of the tracking can take some time so that data will lag a bit. This has to be tested directly with the equipment and adjusted if needed (and of course, if possible).

The following table will show some of the tasks of this project and the approximated difficulty:

<table>
<thead>
<tr>
<th>Component</th>
<th>Difficulty</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collision detection</td>
<td>DIFFICULT</td>
</tr>
<tr>
<td>Implement motion capturing (in real time)</td>
<td>DIFFICULT</td>
</tr>
<tr>
<td>Implement head tracking (in real time)</td>
<td>DIFFICULT</td>
</tr>
<tr>
<td>Adding all components together</td>
<td>DIFFICULT</td>
</tr>
<tr>
<td>Processing the motion data in the control program</td>
<td>MEDIUM</td>
</tr>
<tr>
<td>Creating the VE</td>
<td>MEDIUM</td>
</tr>
<tr>
<td>Setting up the tracking system</td>
<td>MEDIUM</td>
</tr>
<tr>
<td>Setting up the projection system</td>
<td>EASY</td>
</tr>
</tbody>
</table>

Table 2.1: The system components and its difficulty

2.4 Conclusion and discussion

To write a program, which controls all the components of this project, can be pretty complex. The basic programming in C++ wouldn’t be a big problem because there is much help from literature and the internet.

The capturing of the movement is another big part in this project. But the equipment is already set up in the lab and ready to use. Only the calibration has to be done. Attention must be paid to the connection and the software interfaces between the motion capturing computer and the control computer (taking care of the latency).
The tracking system will be used for two components of the project. The first is to capture the movement of the player to position the avatar in the VE. The other component is for the head tracking. With this possibility, the user is able to look around in the scene. Also the scene will be updated according to his position and angle of view.

To get the visual output, a scene graph system called OpenSceneGraph is used. It’s written in C++ and can directly be included into the control program. This is a great advantage because it can be programmed very close to the collision detection component and the tracking client.

The created scene graph will be displayed with a 3D projection system. This passive stereo system will give the user a better feeling of being in the scene. Also this is already set up and maybe just needs some adjustments.
Conception and design

The third chapter will clarify how the project will be realized. With the knowledge from the previous chapter, the requirements and the design can be defined to prepare the implementation of the program. Also the set up of the camera and the placement of the markers will be investigated.

3.1 Requirements

Now with the knowledge of the techniques and methodologies for the components, a clearer picture of this project can be made. In this section, the components of this project are described in a more detailed way. This descriptions lead to the requirements for this project. The requirements will represent the base for the conception and the planning process which are described in the next sections.

These are the main components discovered from the literature review:

- For this project, a tracking system is crucial. This is necessary to capture the motion of the user and the head tracking.
- Also a 3D projection system is needed to give the whole scene more immersion.
- A graphical component will create the VE. Also the obstacles and player representations are build with this component.
- The collision detection system will identify intersections between obstacles and the player representation.
- A control program has to combine and manage all software components.
Also necessary is a fast computer where the control program is running on. It should be a fast computer because it has to do all the calculations and visual renderings. Also the tracking system and the projectors have to be connected to this computer.

Furthermore, a attachment mechanism for the marker to the user has to be developed. The best option is to build it in this way, that it could be attached and removed very fast. Then players can change the marker quite quickly.

3.2 The tracking system

The tracking system, that is being used is an optical tracking system. The cameras are assembled by the company Qualisys [webi] and belong to the Opus 300 series. The system, which is available in the lab, consists of 5 cameras. These cameras are connected to each other where one camera acts as master and is also connected to a computer which runs the tracking software QTM.

![Motion capturing cameras](image1)

Figure 3.1: Motion capturing cameras

These cameras are a bit different than normal cameras and specially designed for motion capturing. The main difference is, that they are emitting infra red light (which is not visible to the human eye). Also, the cameras have filters in front of the lens to reduce irritation by other light. Figure 3.1 is showing an exploded view of a camera. The front glass plate is the light filter. Behind that is the camera lens which is connected to the hardware. Around the lens are the infra red diodes. Furthermore, these cameras are high speed cameras which can capture 500fps, at reduces resolution up to 10000fps [Qua08].

![Marker](image2)

The emitted infra red light is reflected by the marker (figure 3.2) which are made of a highly reflective material. They are called passive marker because they are not emitting light by itself. The markers are available in different sizes and can be attached to the body of a person or to other objects.
To determine a marker in 3D space, at least two cameras have to see the marker. This is enough because the cameras “know” where they are. This is established by the calibration process. The calibration is done by placing a special frame in the scene which can be seen by all cameras. Then a stick with markers attached has to be moved in front of this frame for 20 seconds. After that, all cameras know where they are and the calibration frame can be removed. Now the system is ready to be used and normal markers can be tracked.

3.2.1 The tracking software

The tracking software is a special software, which is connected to the tracking system (the cameras) to receive, process and route data. The software is receiving the data from the master camera and uses it to calculate the 3D coordinates of the markers. The software can also record and play back movement.

The software is running on a separate computer which is connected to a network to transmit the 3D data to other clients. The software is also provided by Qualisys and is called Qualisys Track Manager (QTM) (version 2.4).

The screen shot shows a set of markers from 5 different angles. This software is also able to recognize specific marker patterns (labelled trajectories). With that, it is possible to identify certain parts of the body for example. These marker patterns are called Automated Identification of Markers (AIM) models. The models can be created directly in the tracking software. In order to realize that, one session with the the pattern has to be recorded. All the recorded markers can now be labelled (give them names). When all markers are labelled, the model can be saved.
To use the model, it has to be loaded into the tracking software. Whenever the specific marker pattern appear in the scene, the software will recognize the pattern and matches the model onto it. Now the markers can be identified in real time. This is an essential part in order to make this project work.

After the model for the markers is created, the data has to be transmitted to the control computer. This is quite easy because the tracking software also acts as a server which can be accessed with a special client. This client is going to be explained later.

### 3.2.2 Placement of the markers

The placement of the markers is important and should be considered in some aspects. The markers have to be visible to the cameras all time. Otherwise the tracking has gabs or doesn’t work at all. For this project it is required, that the highest and the lowest point of a person is tracked. That means the head and the feet.

This is enough to create a very basic virtual character which would be a box for example where the head is the top edge and the feet the bottom edge.

In later development stages, this design can be improved to a more sophisticated level where the legs are separately displayed and the arms are included. But for first tests, one box should be enough.

In order to get a properly working system, both feet have to be processed. That means, that always the lowest foot represents the bottom edge of the box. The box should then resize with the motion of the player. If he ducks or jumps, the box will get smaller.

So at first, three markers are enough to get this functionality (see figure 3.4). One on each foot and one for the head. The markers on the feet could be attached to the shoes and the one for the head can be attached to the glasses which are used anyway for the 3D system.

When the markers are attached, a AIM model has to be created and included to the tracking software to identify the markers. How this is done was described in section 3.2.1.

For the glasses, a marker pattern would probably be the better solution instead of only one marker. This would improve the stability of the head tracking. The project would work with only one marker for the head but it is recommended to use more markers (three would be good).
3. Conception and Design

3.2. The Tracking System

After that, the tracking system is ready to track the head and the feet in real time.

3.2.3 Placement of the cameras

The placement of the tracking cameras plays a very important role in this project. Because the player is moving a lot in a quite big area, the cameras have to cover a lot of space. For that reason, the arrangement has to be very sophisticated to track all markers at any time.

In the lab are five cameras available. Because one marker has to be seen by at least two cameras, the best arrangement would be, that two cameras are tracking the head, two cameras are tracking the feet and one camera is trying to track a bit of both. In that way most of the space should be covered. It is also important to pay attention to the movement of the player. It can happen quite easily, that the player covers a marker when he moves or when he ducks.

In that case some cameras have to be close to the ground to track the feet correctly.

![Figure 3.5: Photo of the camera setup](image)

This arrangement should be the best one but it has to be tested. If the tracking seems to be unstable, the arrangement has to be changed again. The best solution would be to get some more cameras. This would help to cover a greater area and avoid hiding marker while jumping, ducking and walking around.
3.3 The projection system

The projection system, that is going to be used consists of two projectors from the company Christie. The model is the LX45 which are very bright and have a high contrast. Like described before (see section 2.2.5), the devices are projecting on the same canvas and emit polarized light to get the 3D effect which can be seen with the special glasses.

The projectors are mounted to the ceiling as a front projection system. This could be problematic because the user can obscure the screen and hide parts of the scene. That could be irritating while using the system. A back projection system would definitely be the better choice but it is not available in the lab. This would also take too much space in this room.

The projectors are connected with the computer where the the VE is rendered and displayed on. They are connected via the Digital Visual Interface (DVI) port. Only attention has to be payed to the driver settings. They have to be set to clone mode for this application. After that, the projector system is ready to be used.

3.4 The control software

The control software has to be developed to manage all components of this project. It is the centre of all information flow and the heart of this project. All functionalities of the game are implemented here. This will be the most crucial and difficult part to provide for this project.

The control software consists basically of this parts:

- The client for the motion capturing
- The graphical component (OpenSceneGraph (OSG))
- All the classes and functions for the game

The task of the control software is basically to connect the motion data with the graphical component and to manage all actions, that are necessary for the game. All parts of the software are written in C++. The reason for that is, because OSG is based on C++ as well as the client for the tracking system.

C++ is a very powerful language and a lot of attention must be payed while programming not to create faulty code which could crash the program.
3.4.1 Integration of the motion data

First thing to integrate in the control software is the motion data from the motion capturing cameras. Like described before (see section 3.2.1), the tracking software is acting as a server which transmits all the data to the clients.

The task is to create such a client. The company Qualisys delivered the system also with a test client, written in C++. This client could be used and adjusted for this project.

![Figure 3.7: Motion capturing server and the client](image)

The client is connecting to the server with a socket connection through the network. This requires that both computers are in the same network. The connection itself is a TCP/IP connection over a predefined port (which is 22222 as default).

The test client basically consists of a Connect() and Disconnect() function to enable and disable a link to the server. Furthermore, there is a function to send commands to the server (SendCommand()). With this commands you can define, which data you want from the server. The function RecvMessage() catches the data coming from the server and pushes it into a data variable. This data variable is processed in the function PrintData(). In this function, the 3D data is printed to the console. See the following chart (figure 3.8) for the information flow.

![Figure 3.8: Flow chart of the test client](image)

Basically, this test client can be used for this project the way it is right now. Only the function PrintData() has to be adjusted to fit into the program. This function will process the data.
for the VE. Also some parameters have to be adjusted to get the correct data from the server. What these parameters are exactly and what else will be edited in the test client, will be described in the implementation section later in this document.

Basically, what the client has to do is getting the data of the foot markers for the movement of the player box and the head marker for the head tracking.

### 3.4.2 OpenSceneGraph

OpenSceneGraph, the graphical component in this program, will be a very big part. Its purpose is to calculate and show all the visual elements of the game.

To get a overview of all the basic visual elements, a modified version of figure 2.2 is shown:

![Figure 3.9: Example screen with descriptions](image)

Let's start with the frame around the scene. Most important is the plane on the ground where the player is walking on. This plane is limited to the left and right that the user can not walk out of the scene. The planes around the scene (left, right and top) helps to position yourself and makes the head tracking look more realistic.

One thing to consider about this frame is to make it solid or a wire frame. A solid one can be narrowing while playing, so maybe a grid would be the better option.

This frame is a static element so it does not move when the player is walking around.

The obstacles are boxes in different sizes and at different positions. They will be generated far away inside the frame and will move with different speed towards the player.

The size and the position of the obstacles could be generated randomly. This will create a lot varying shapes. The problem is, that the level of difficulty can not be controlled very easily.

Another approach would be to create a set of pre defined obstacles. In that way, the designer could choose different obstacles for diverse difficulty levels. This would be more flexible and better to handle.

The movement of the boxes have to be animated. This could be done by decreasing the distance to the player in the main draw loop “by hand” or by using a AnimationPathCallback object. With this object, you just need to say from where to where in what time the box should
move. The movement itself is done by a separate thread and you don’t have to take care about that. This would be the best solution.

The third element in the screen (figure 3.9) is the player box. This box is steered by the player, so the position is defined by the motion data (see section 3.4.1). Also the height of the box will adjust with the movement of the player (when ducking, jumping, ...). The player box is also half transparent to avoid covering coming obstacles.

**Collision detection**

The collision detection is a part of OpenSceneGraph. OSG has the functionality to use bounding boxes (not visible), which are used to test intersections between two or more bounding boxes. OSG already has a intersection test implemented so there is no need to write an own function. Like described in the literature review (see 2.2.2), this intersection function has to be tested, if it is applicable for this application.

So every obstacle box and the player box has to have a bounding box and the intersection test has to be applied all the time to detect collisions. Important is, that the test only applies for the player box and the obstacles and not for the obstacles among each other.
Implementation

This chapter will explain, how the system has been implemented. All classes and important functions will be presented and explained. Furthermore, the scene graph will be described and how the obstacle management works. Also the obstacle handling will be described.

4.1 Introduction

The software is developed on Microsoft Windows machines. The main development computer is running with Windows XP, the second one with Windows 7.

Like mentioned before, the system will be programmed in C++. For that, the Integrated Development Environments (IDEs) Microsoft Visual Studio 2008 and Microsoft Visual C++ Express 2008 are used. This programs are used because the libraries of OpenSceneGraph are compiled especially for this software.

Also some adjustments in the project settings have to be done. Most important is to include all libraries, that are necessary for the project and to tell the compiler where the includes, libraries and binaries are. After that the IDE is ready to use.

The libraries of OpenSceneGraph are installed in version 2.8.2 and are implemented in the mentioned IDEs to compile the source files. Also some environment variables in Windows have to be set to compile without errors.

For the tracking client, an additional library is needed, which provides the network connections to the server. But this is a standard library, which doesn’t have to be downloaded.

After setting up all these properties, the development process can start.
Global constants
First of all, this program is using a header file, which is included by every other source file in this project. It has been created to provide one central point, where global constants can be defined and maintained.

This simplifies adjustments a lot, when the same attribute is used in several source files. Only one modification has to be done in order to adjust this parameter in the whole program.

![Diagram showing constants](image)

**Figure 4.1:** The first few constants of the CONSTANTS.h

These constants are quite a lot (more than 80) and they are not going to be explained in this document because it would take too much space and it is not relevant, to understand how the program works. They are described directly in the source file (CONSTANTS.h).

The diagram (figure 4.1) is showing just some of the constants. They are basically used to calibrate the tracking system, define constraints where the player can walk, light attributes, object names and presets of the obstacles.

Also the difficulty levels are defined here. This affects the generation rate and the shape of the obstacle. However, this is going to be explained more in detail later in chapter (section 4.5).

### 4.2 The main function

The main() function is the heart and the point of entry of the control program. Most of the initialization (camera, lights, shadows, menu, ...) is done in this function.

Also the tracking thread gets started here, which establishes the connection to the tracking server.

This function also contains the main drawing loop. The main drawing loop is a infinite while loop and is part of OpenSceneGraph. It is refreshing the scene with every cycle and can be terminated with setting a parameter.

In the loop are the obstacle generation / deletion, the collision detection and the update for the labels. This loop also calls the function to render a new frame.

The following diagram (figure 4.2) is showing the main() function and the belonging functions and parameters.
The main() function also contains the root node of the scene graph. The top node is the viewer. The viewer contains all information about the camera, event handlers, scene data and all the parameters of the scene. The scene data is the root node of the scene graph. In this case, it is the object root. All further objects of the scene are added to this root object.

Next, the functions of the main class are going to be explained:

- **drawPlane()**
  This function returns the plane, where the user is walking on.

- **initHUD()**
  The HUD is a small info output (only used for debug purposes) and is initialized by this function.

- **initLights()**
  This function initializes the light. This is necessary for the shadows.

- **initShadows()**
  This function sets the shadows parameters for specific objects (the obstacles and the plane). Shadows are only used for the “catch” mode.

- **initSupportElements()**
  The support elements are graphical objects, which are added to the scene (the grid, axes and frame). This is also used for debug and not visible while playing, except for the frame.
• **initViewer()**
  In this function, all the parameters for the viewer (like camera, event handler, display settings) are set.

• **main()**
  This is the main() function which is initializing the system and contains the main drawing loop. This function is the entering point of the program.

• **resetGame(unsigned int)**
  At the end of the game, some parameters needs to be reset. This is done by this function.

• **RTThreadRoutine(LPVOID)**
  This is the thread routine to start the tracking client. It calls the function `startTracking()` from the RTClient (see section 4.3).

The following chart (figure 4.3) explains the information flow. Due to the fact, that the main() function is the centre of this program, it has many links to other classes and functions which are not displayed in this diagram.

The other diagram (figure 4.4) is explaining the main drawing loop a bit more in detail. Every action is executed once in one cycle.
4. IMPLEMENTATION

4.2. THE MAIN FUNCTION

Figure 4.4: Tasks in the main drawing loop
4.3 Implementation of the tracking client

The tracking client for this project can be adapted from the test client described in section 3.4.1. Some small adjustments have to be made to achieve the required compatibility for this project. Most important is the fact, to let the tracking run in an own thread. This is necessary to avoid problems with the main drawing loop for the visuals. Therefore, the `RTThreadRoutine` has been created in the `main()` function.

4.3.1 The RTClient

The `RTClient` (realtime tracking client) is the modified version of the test client from Qualisys. The main difference is, that its `main()` function is renamed to `startTracking()`. This is necessary, because this client is not an independent program anymore and doesn’t need a point of entry. The function `startTracking()` will start the tracking and is called from the tracking thread, which is started from the `main()` function of the program.

![RTClient Diagram]

The class diagram (figure 4.5) is showing all functions of the `RTClient`.

The following functions are staying the same as in the test client. Only the function names are adjusted.

- `connect(CStdString, int)`
  Here, a connection to the server is established (it creates a socket).

- `rtDisconnect()`
  This function disconnects from the server.

- `recvMessage(SHeader&, CStdString&)`
  This function puts the data, coming from the server, into a string object.
• sendCommand(CStdString)
  To send a command to the server, saying which data the server should send, this function
  is used.

There will be no further descriptions for these functions because they are not developed for this
project. They are used like in the test client.
The only thing to consider is to adjust the command for the server. This command should ask
for labelled makers. In order to do that, the string should be changed to SendCurrentFrame
3d Analog. With this request, the server will reply 3D data of the labelled markers.

The new and adapted functions are:

• processData(CStdString, std::vector<CStdStringA>)
  This function is called from a infinite loop and delivers the data packages. It routes the
data to the head tracking and the box movement function which are explained next.

• processMultipleMarker(S3dComponent*,
  std::vector<CStdStringA>)
  This function processes the data of the feet to position the player box in the scene. For
that, the centre of both feet is calculated to get the position. Also the height of the box is
calculated. This is done by getting the lowest foot and the head. Between this markers,
the box is created. The position and the height of the box is adjusted all the time while the
tracking is active.

• startTracking(Player*, osg::ref_ptr<osgViewer::Viewer>)
  The start function of the tracking. It initializes the tracking system and establishes the
connection to the server. Also the loop to receive the data is located in this function.

• trackHead(S3dComponent*, std::vector<CStdStringA>)
  The head tracking only uses the data from the head markers. Depending on the position
of the head in the scene, the camera is moved and rotated to get the feeling that you are
walking in the scene. So if you are moving your head to the left edge, you are also seeing
the scene from the left.

To update the scene and the player box, a pointer to the main() function is established. With
the help of this pointer, all objects in the scene graph are accessible. So the objects can be
updated immediately without returning any value from a function.

The function processMultipleMarker() calculates the position of the box. It also has
a logic implemented to keep calculating, when one foot marker is lost. Then, the last known
position is taken for the computations.

The following flow chart (figure 4.6) shows, how the RTClient (and its thread) works. Again,
it is important to know, that the RTClient runs in a separate thread. In this case, it runs parallel
to the main() function, which gives more performance to the system.
The function `rtDisconnect()` is part of the RTClient, but is called from the `main()` function when the program gets terminated.

This class is also using a set of structs to store the different data coming from the server. For example, there are structs for non-labelled markers, 3D labelled markers or 6D labelled markers.

### 4.3.2 Additional components

The RTClient also uses some additional components which are mentioned here. They are also provided by Qualisys and not developed or adjusted for this project.

**Class StdString**

The class `StdString` is a string template and is used in the test client for storing the tracking data. The test client uses this type instead of the standard string type (`std::string`). It is used because it has special features for casting and reading from / writing to streams which is necessary for the server-client communication.

It could probably be replaced by the standard string type but this requires some deep changes in the basic RTClient functions. The usage of the StdString works fine so there is no need to change that.
Class NBC_Markup

The NBC_Markup class is basically a Extensible Markup Language (XML) parser. The usage is necessary because the data coming from the server is a XML stream. The parser is used to get the information out of this stream.

It would also be possible to use string comparisons and get the data out by hand but this is very fragile for errors and probably pretty slow. This parser delivers functions to get to the wanted information inside the stream.

This code listing shows a fragment of a XML stream which is coming from the server. For example, the function FindChildElem() can be used to jump inside a searched tag. The function GetChildData() delivers the value of this tag.

In the end, the application of this parser is pretty useful and handy. It saves a lot of work and time (meaning performance) to traverse though all the XML elements and to extract their values.

It was used in the test client and will be implemented in this project as well.

```
<QTM_Parameters_Ver_1.00>
  <The_6D>
    <Bodies>3</Bodies>
    <Body>
      <Name>measurement frame</Name>
      <RGBColor>255</RGBColor>
      <Point>
        <X>505.900000</X>
        <Y>445.516705</Y>
        <Z>-689.827939</Z>
      </Point>
      <Point>
        <X>342.204156</X>
        <Y>933.915600</Y>
        <Z>-535.076816</Z>
      </Point>
      <Point>
        <X>693.550365</X>
        <Y>506.866771</Y>
        <Z>-684.991285</Z>
      </Point>
      <Point>
        <X>1209.655119</X>
        <Y>673.697967</Y>
        <Z>-673.144574</Z>
      </Point>
    </Body>
  </The_6D>
</QTM_Parameters_Ver_1.00>
```

Listing 4.1: XML stream fragment from the server
4.4 Implementation of OpenSceneGraph

The next classes, that are going to be explained, are based on classes and functions of OpenSceneGraph and provide the basic functionalities for the game. Also some helper classes are going to be explained, which helped in the developing process.

![Class diagram of OpenSceneGraph related classes](image)

Figure 4.7 is showing the classes, that are related to OpenSceneGraph and how they are connected.

The centre is the main class. Each component is connected there. The Menu, Player and Obstacle are the basic classes which are a crucial part of the game. Utilities and KeyboardEventHandler are helper classes to provide alternative test functions and controls.

4.4.1 The basic classes

The basic classes are the key classes, which are directly necessary for the game (in contrast to the helper classes, which are explained later in section 4.4.2).

**Class Obstacle**

The class Obstacle represents an obstacle in the game. It consists of a visual element (the box itself) and attributes and functions to control the obstacles (like movement, deletion, ...).

The attributes of an obstacle are an obstacle-structure, which is the size, the bounding box and the box itself. Furthermore it has an attribute to do the animation (AnimationPathCallback) and a node where the object is in. This node will be added into the tree in the main() function (see section 4.2).
An obstacle is quite a big object. It has some attributes and functions to control its behaviour. Furthermore, the obstacles have a function, which controls the deletion. If the obstacle moved behind a certain point, it gets deleted.

- **Obstacle(int)**
  The constructor of this class takes a difficulty attribute to generate an obstacle according to the difficulty level. It initializes all objects of the obstacle and starts the animation.

- **generateRandPosForCatch()**
  This function generates a random spawn position for the obstacles in catching mode. This function is called from the constructor.

- **generateRandSize()**
  This function was used to create a random-size obstacle. But it became obsolete in the implementation process. However, it is now just used for test purposes.

- **updateBoundingBoxes()**
  This is the update function for the bounding boxes. The new values are calculated from the position of the obstacle boxes.

- **getCrashed()**
  This function returns true, if the obstacle already collided with the player box. This is necessary to avoid getting multiple hits while intersecting.
• getRandColor()
  The obstacles have different colours. This function is called from the constructor and
returns a random colour.

• getRandObstacle(int)
  This function returns a random obstacle according to the difficulty and only applies for
the normal play mode. This function is called from the constructor.

• getTrans()
  This returns the node of the obstacle. This function is called from the main() function
to add the obstacles to the root node.

• obstacleFade()
  The purpose of this function is to let the obstacles appear slowly in the scene. It avoids a
hard occurrence.

• obstacleOutOfField()
  This function returns true, if the obstacle moved out of the playing area. If this is the case,
the obstacle can be deleted (done in the main drawing loop).

• setCrashed(bool)
  This function sets the crashed attribute on the first intersection of its obstacle and the
player box.

• getBoundingBoxes()
  To get the bounding boxes of an obstacle, this function is called. It returns an array because
one obstacle can have more than one bounding box.

The class Obstacle is designed in a way, that it can be build up out of more than one box. This
also requires more bounding boxes for the intersection test. The obstacles are more explained in
a upcoming section (4.5) in this document.

**Class Player**

The class Player is derived from the Obstacle class. Actually, the player box is also an
obstacle. Because of that, most of the functions of the class Obstacle can be used for the
player box as well. Only some player specific attributes and functions are added.
4. IMPLEMENTATION

4.4. IMPLEMENTATION OF OpenSceneGraph

Figure 4.9: Attributes and function of class Player

Important is here the `testIntersection()` function. It is called with every new frame from the `main()` function to test a collision between the player box and an obstacle.

- **Player(osg::Ved3f, osg::Ved3f)**
  The constructor of the player sets all initial attributes. There will be only one instance of this player in the game. This instance is created in the initialization of the `main()` function.

- **initFlashScreen()**
  This is the initialization of the flash screen, which indicates a collision with an obstacle. It is called in the start up sequence in the `main()` function.

- **setFlashLabel(bool, bool, char*, char*, bool)**
  This function sets the overlay to let it flash. With the parameters it can be defined if the screen, the player box or both should flash.

- **setIdle()**
  This function sets the colour of the player box to grey. This indicates, that no game is started and the player is in idle mode.

- **setSize(osg::Vec3d)**
  Updates the height of the player box. The function is called from the `RTClient` which is calculating the height from the data, coming from the tracking cameras.

- **testIntersection(Obstacle*, unsigned int)**
  This is the intersection test of the player box with an obstacle. It is done with help of the bounding boxes and is called in every cycle of the main drawing loop. It returns true, if a collision happened and updates the indication labels.

- **updateBoundingBoxes()**
  Like in the `Obstacle class`, this function updates the bounding boxes. It is an adjusted version for the `Player` class.
The dimensions of the player box are always the same, except for the height, which is variable. The size of the width and depth are defined in the `CONSTANTS` header.

**Class Menu**

The class `Menu` is responsible for the menu bar on the top of the application. It creates the buttons and assigns actions to them. It also implements the mouse click handler. Furthermore, the start sequence and the game over animation are located in this class. Both of these animations are called from the `main()` function.

The animations are started on certain conditions of the game. For example, the start animation will start, when the user clicked on a button to start the game. The game over animation will show up, when the lifes of the player are depleted.

- **Menu()**
  The standard constructor of this class initializes the attributes and the font styles.

- **clickedOnExit()**
  If the player clicked on the “Exit” button, the return value will switch to true.

- **clickedOnStart()**
  If the player clicked on a difficulty, this function will return true and the game starts.
4.4. Implementation of OpenSceneGraph

- `countAnimOver()`
  This function returns true, if the count down animation is over and the game can begin.

- `createMenuItem(const char*, char*, Vec3f, float)`
  This function returns a button for the menu bar with a name, texture, position and size.

- `gameOverAnimationOver(bool)`
  Test to see, if the game over animation is over. If the animation is over, the screen will disappear.

- `getDifficulty()`
  To get the chosen difficulty, this function is used. It is called from the `main()` function and is needed to provide the correct obstacles for the different levels.

- `getMenuLayerObject()`
  This function returns the node, in which the menu is stored. The `main()` function calls this function and pushes the node into the root node of the scene graph.

- `getMenuSpacer(osg::Vec3f)`
  This will return a spacer for the menu bar.

- `handle(const osgGA::GUIEventAdapter&, osgGA::GUIActionAdapter&)`
  Because this class is derived from the `GUIEventHandler`, this function has to be overwritten to get the mouse click events.

- `initCountDown()`
  This function initializes the count down animation at the beginning of the game.

- `initGameOverScreen()`
  This function initializes the game over screen, which occurs at the end of the game.

- `initMenu()`
  The menu layout is initialized here.

- `pick(osgViewer::View*, const osgGA::GUIEventAdapter&)`
  This function is performing the actions which are triggered by the `handle` function. It is checking which button was clicked and runs the associated action.

- `reset()`
  To start a new game, some attributes have to be reset. This is achieved with this function.

- `setHitCounterLabel(std::string, osg::Vec3f)`
  This function updates the hit counter label on the top left of the screen and shows the score at the end of the game.
4. IMPLEMENTATION

4.4. IMPLEMENTATION OF OpenSceneGraph

- `showGameOverScreen()`
  When the game is over, an animation will be shown.

When the user has chosen a difficulty level, an internal variable is set. This variable will be carried
to the `main` function, where it adjusts the spawn time and the shape of the obstacles.

The `main()` function updates the counter label (in the left top corner) with help of the function
`setHitCounterLabel()`.

4.4.2 Helper classes

The helper classes are used to simplify the implementation and testing process. They don’t actually participate in the game.
It can be seen as a set of tools to help building and test parts of this program. This tools enable the programmer to get specific information from the program while it is running or to change internal values to see how the program responds and if the workflow is correct and without errors.

**Class KeyboardEventHandler**

The `KeyboardEventHandler` is a class to catch and process key inputs. This class is not used by the game and was implemented to move the box and the camera with the keyboard when the tracking system is not available.
Thereby, it was possible to test the collision system and to see if the objects are placed correctly for example, without using the tracking system. Also, the game could be simulated in an easy way.

To get the key input, the class `GUIEventHandler` had to be extended. This base class is part of OSG.

![Figure 4.11: Attributes and functions of class KeyboardEventHandler](image)

The functions in this class are overwritten from the derived class `GUIEventHandler`. This class does not have any further functions.
The following table is a compilation of implemented keys and their actions.

<table>
<thead>
<tr>
<th>key</th>
<th>action</th>
<th>key</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1</td>
<td>show OSG info screen</td>
<td>w</td>
<td>move the camera forward</td>
</tr>
<tr>
<td>F2</td>
<td>print OSG info to the console</td>
<td>s</td>
<td>move the camera backward</td>
</tr>
<tr>
<td>F3</td>
<td>show grid</td>
<td>a</td>
<td>move the camera left</td>
</tr>
<tr>
<td>F4</td>
<td>show axes</td>
<td>d</td>
<td>move the camera right</td>
</tr>
<tr>
<td>F5</td>
<td>show wire frame</td>
<td>t</td>
<td>rotate the camera up</td>
</tr>
<tr>
<td>F6</td>
<td>show wire frame</td>
<td>g</td>
<td>rotate the camera down</td>
</tr>
<tr>
<td></td>
<td></td>
<td>f</td>
<td>rotate the camera left</td>
</tr>
<tr>
<td></td>
<td></td>
<td>j</td>
<td>rotate the camera right</td>
</tr>
<tr>
<td></td>
<td></td>
<td>+</td>
<td>move player box up</td>
</tr>
<tr>
<td></td>
<td></td>
<td>#</td>
<td>move player box down</td>
</tr>
<tr>
<td>arrow up</td>
<td>move the player box forward</td>
<td>Return</td>
<td>activate / deactivate player box</td>
</tr>
<tr>
<td>arrow right</td>
<td>move the player box right</td>
<td></td>
<td></td>
</tr>
<tr>
<td>arrow left</td>
<td>move the player box left</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4.1: Keys and its actions

The player box and camera movement does not work when the tracking is active. The inputs get overwritten by the tracking data. What works are the F keys and the return key. If the tracking is switched off, the other keys can be used to move the player box and the camera around. But these controls are just for debugging and not intended to be used in the while the game is running normally.

Actually, the keyboard controls are not needed at all. In later developments, it could be removed. The settings for the grid, the axes and so on could be adjusted with a proper settings menu. The same holds for activating / deactivating the player box.

**Class Utilities**

The class Utilities is basically to provide functions which doesn’t have directly something to do with the game. Most of them are for debug purposes like printing data to the console or to the screen.

The following diagram (figure 4.12) is showing all utility functions.
The class `Utilities` is included in all other classes, where its functions are needed. For example, the function `findNamedNode()` is used in some classes, to get a node by its name. Also the `print-` functions are used quite a lot.

- **addAxis()**
  To see, where the axes are, three lines (x, y, z) are placed in the centre of the scene. This is not shown while the game is running. It can be displayed with the key F4.

- **addCage()**
  This function returns the wire frame around the whole scene.

- **addGrid(int, int, int, int)**
  This will lay a grid on the ground. It is useful to determine, where objects are in the scene. The grid is not shown while the game is running. It can be activated with the key F3.

- **createBox(osg::Vec3f)**
  This function returns a normal box with specific dimensions. It is useful because it takes some lines to create a simple box.

- **createHUD(int, int)**
  The HUD is a graphical element, where debug information can be added. This is not shown in the normal game. It can be activated with the key F6.

- **findNamedNode(const char*, osg::ref_ptr<osg::Node>)**
  This function returns a node from a tree. Important is that the node has a name. Then you can search for that name in a given tree.

- **printError(char*, bool, bool)**
  Prints an error message to the console. It also has the option to pause or to terminate the program.
• printInfo(char*, bool)
  Prints an info message to the console. Again, this function has to option to pause the
  program.

• setLabelText(const char*, std::string)
  This function updates the text on the HUD. If the label is not there yet, it will be created.

### 4.4.3 Scene graph chart

The scene graph is a hierarchy representation of all visual elements in the scene. OpenSceneGraph organizes the objects (called nodes), like the name already says, as a tree structure. That means, that every node in the scene has a parent node and can have child nodes, if it is not a leaf.

The diagram (figure 4.13) shows the nodes in this project and how they are organized in the tree. The viewer holds the scene data. The scene data is a compilation of all objects, that are visible in the scene.

The viewer can hold one node, which is the root node. The root node contains all other objects.

Important to know for this graph is, that the obstacles are added to the shadowed scene if the mode “catch” is activated. This will show shadows of the obstacles on the plane and enables the player to estimate the position of the obstacle better.

In normal mode, the obstacles are added directly to the root node. In this case they don’t have shadows because they are a bit confusing and are causing some problems with displaying the obstacle.

This graph is quite small but the obstacles (which is just one point in the graph) are a big part in the tree. There are always multiple instances in the scene and they can be compiled of several boxes itself.

![Diagram of the scene graph](image-url)
4.5 The obstacles

This section of the implementation explains, how the obstacles are realized. All obstacles are predefined shapes and can consist of more than one box. This option allows the designer to create a lot of different shapes. Every box of the obstacle has to have an own bounding box. So the intersection test checks every single bounding box of the obstacle.

The presets also allow to use defined obstacles for diverse difficulties. For example, only obstacles 1, 3, 4 and 7 are used for the difficulty “easy”.

Figure 4.14 is a compilation of all obstacles that are implemented in the game. All obstacles are created from at least one cube. The edges in some figures indicate, how the cubes are arranged to build up a obstacle.

The words under the obstacles (easy, medium and hard) indicate, in which difficulty level the obstacle has been implemented. So the different difficulty levels have separate obstacles where an obstacle can be used for more than one difficulty level as well.

Also the spawn rate (the frequency, where the obstacles are created) is different:

- easy: new obstacle all 3 seconds
- medium: new obstacle all 2 seconds
- hard: new obstacle all 1.5 seconds

These two settings are made, when the player chooses a difficulty level. This level will stay the same till the game is over. After that, the player can choose another level.
Figure 4.14: The implemented obstacles
CHAPTER 5

Results and evaluation

This chapter will describe tests and results of the project. In order to do that, several test subjects are using the system and are giving feedback about their experiences. This results will be presented and analysed. The gained information is used to fix errors and to improve the program.

5.1 Introduction

The description of the implementation (chapter 4) is the result of two prototypes. Only the final version has been described. In this chapter, the process of getting to the final version is explained a bit more in detail.

After implementing the system, it would be interesting to know, how useful and usable it is. For that reason, some participants have been asked to test both prototypes and to provide feedback with the help of questionnaires.

The development process of the program was a combination of checklists and evolutionary prototyping [RB92].

The first prototype was build with help of a checklist. This checklist was compiled as a rough plan before programming and got refined in the development process. This was very helpful to sub-divide big parts into small tasks. It also helped to handle and organize small and less important tasks.

This type of task management was used till the end of the project. An idea could be defined as tasks on that list immediately.

Also evolutionary prototyping was used for this project. That means, that a prototype gets evolved over and over again. Also feedback from testers has been incorporated in the process.
5. RESULTS AND EVALUATION

5.2 USAGE AND INSTRUCTIONS

Figure 5.1 is showing the process of the software development. First of all, the problem has to be analysed. This leads to the requirements. After that, the system can be designed and implemented. The testing and evaluation is showing, how good the program was implemented (in terms of correctness and usability). With the feedback of the testing and evaluation, the system can be analysed again with the gathered information. Now the process starts over again. Doing this over a period of time, this process is called evolutionary prototyping.

5.2 USAGE AND INSTRUCTIONS

The usage of the program is very simple. The player just have to wear the prepared glasses with the markers and the markers on the feet. Now he can be tracked in front of the screen.

Figure 5.2: The markers for the feet and the prepared glasses
5. RESULTS AND EVALUATION

5.2. USAGE AND INSTRUCTIONS

Figure 5.3: A player equipped with markers on the feet and the head

After the player is ready, the tracking cameras and the server have to be switched on. The server should be configured with the marker model “glasses&feet”.

Figure 5.4: Screen shot of the Qualisys tracking manager with loaded model
The labelled trajectories, which can be seen on screen shot 5.4, are now matching the marker pattern, that is currently visible. The screen shot also shows the scene in 3D. The blue markers are the glasses, the green ones are the feet. Also the position of the cameras is displayed.

When all settings are made and the markers are attached, the player can start the game by clicking on a difficulty. After a count down animation, the obstacles start their way towards the player. Since prototype II, the player can see his score and the lifes he has left in the top left corner.

5.3 The first prototype

The first prototype was finished after about two months of work. It was created according to the conceptual design without any influence from test subjects. This release should prove how good the system works for other people than the designer and should give information about the usability.

The feedback from the test subjects should shed light on further improvements and optimizations. With this information, the next prototype is created.

5.3.1 Screen shots

The following pictures are showing the first prototype in action. The screen shots are captured from the screen where the 3D effect is not activated. A photo of the canvas with activated 3D, would shows a blurry picture because the photo camera can not focus on two images separately. The screen shots are presenting the most important events in the game.

![Figure 5.5: Initial view](image)

![Figure 5.6: The player box with the start sequence](image)
5. Results and Evaluation

5.3. The first prototype

5.3.2 Evaluation

This section describes the result of the first evaluation. For this, six participants have been asked to play the game and to give feedback. The feedback is provided by a questionnaire, which can be found in the appendix A.1 of this document.

What follows is the result of this evaluation. The answers of all participants have been added to the upcoming diagrams and descriptions. The diagrams are presenting the results to the questions in its caption. The y-axis is showing how many participants ticked the answer on the x-axis.

---

Figure 5.7: Screen shot while playing
Figure 5.8: Screen shot while playing (player is ducking)
Figure 5.9: Player got hit by an obstacle
Figure 5.10: Game over animation
5. Results and Evaluation

5.3. The First Prototype

Figure 5.11: Q.1 How difficult was it to use the program in general?

Figure 5.12: Q.2 Do you think the movement of the player-box matched your movement . . .

Figure 5.13: Q.3 How long did it take you to get the right feeling for the distances to the obstacles?

Figure 5.14: Q.4 How convincing was the impression of 3D?

Figure 5.15: Q.5 Did you have the feeling of “being in the scene”?

Figure 5.16: Q.6 What would you say about the speed of the obstacles?

Figure 5.17: Q.7 Did you have problems with:

- orienting in the scene
- positioning yourself in the scene
- estimating distances
- estimating speed
- moving around

- serious problems
- some problems
- neutral
- no problems
- N/A
5. RESULTS AND EVALUATION

5.3. THE FIRST PROTOTYPE

Q.8 If applicable describe the problems from question 7:

All participants had severe problems with the tracking. It got lost several times while playing. Because of that, the gaming experience got disturbed pretty often and the player had to make sure, that the tracking is active. This is a big problem because the player should be fully focused on the game and not on the tracking.

Q.9 Did you have other problems? If so, please specify:

The own shadow on the screen was a problem because it was hiding an important part of the screen. It was exactly on that spot, where the player box is located. This made it more difficult to estimate the distances from the player box to an obstacle when parts of the screen are hidden.

Q.10 Do you have ideas / suggestions for improvements and features?

Most people asked for different difficulty levels because this first prototype only had one. Also an dynamic difficulty would be interesting, where the speed increases while playing.

Some participants asked for more sophisticated obstacles because the obstacle set in the first prototype was very simple and limited. Furthermore, some colours for the obstacles would be nice because all colours in this game are kind of greyish.

Some people asked for a better indication of the points that have been scored and the lifes that are left. This combined with a highscore table would also be interesting as well as some statistics at the end of the game (like highest jump, fastest speed, longest time without a collision, . . . ).

One person had the idea of a multi player mode where two (or maybe more) player could play against each other at the same time.

Also the menu navigation should be done by the player itself. At the moment, a second person has to click on the buttons or the player has to click and move very fast back to the playing area.

A more detailed player avatar would help a lot to walk in the game. In this case, you could use more realistic movements avoid collisions. Another person had the idea to play completely without the player representation.

Q.11 Do you have any other comments?

It seemed that most of the participants had fun, despite the fact that the tracking was not really good working.
5.3.3 Summary

After the first evaluation, this summary explains the major problems and improvements. This leads to the requirements for the next prototype.

One main problem is, that the program was pretty unstable and crashed several times. The cause is an access violation in the memory which terminates the program immediately. This error could not be solved for the first prototype and needs some extensive investigation for the next release.

This problem is very annoying because it occurred pretty often and forced the player to restart the game. The fix will have high priority for the next prototype.

Another big problem was the tracking system. Due to the fact, that the tracking should work in a big area for head and feet, the camera positions have to be adjusted very well.

This was obviously not the case for the first testing session because the tracking got lost very often (despite the fact, that most people thought the tracking worked pretty good according to figure 5.12, most of them had problems with moving and positioning (figure 5.17)). The cameras could not recognize the model of the markers. That led to a jolting movement of the player box and disturbed the flow of the game a lot. The player was acting in the right way but the game was not responding correctly.

This behaviour has to be fixed in order to make this game more enjoyable.

Question four (figure 5.14) was asking for the 3D effect. The majority of the participants were pretty happy with the result. Probably, some improvements could be applied to make the 3D effect more convincing. One problem is, that the projectors are not adjusted perfectly. They have to be moved slightly that the pictures overlap correctly. Also some experiments with the OpenSceneGraph parameters could help to increase the impression.

Because of the time pressure, this adjustments are categorized as low priority.

Question five is pretty interesting. Most of the participants didn’t really have the feeling of being in the scene. This was planned to be a major part of this project. Some considerations have to be done in order to improve this. One idea, which came from one participant, would be to switch off the player box. In this case, only the head tracking system indicates, where the player is in the scene.

This would definitely give more immersion to the game and should be planned as an option for the next prototype.

Another improvement, that most of the participants asked for, is a highscore and a proper score indication. This also would urge the players to score the most points and to keep playing. The first prototype only shows how many lifess the player has left till the game is over.

Good news is, that the system responds very quickly. So there is no lag of time in the virtual environment. The player box is moving exactly on time.
**5. Results and Evaluation**

**5.4 The second prototype**

The following table is a short compilation of the next major development steps. On base of that requirements, the next prototype is developed.

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fix the access violation error</td>
<td>HIGH PRIORITY</td>
</tr>
<tr>
<td>Fix the tracking system</td>
<td>HIGH PRIORITY</td>
</tr>
<tr>
<td>Implement different levels of difficulty</td>
<td>HIGH PRIORITY</td>
</tr>
<tr>
<td>Implement more obstacles (different shapes and colours)</td>
<td>MEDIUM PRIORITY</td>
</tr>
<tr>
<td>Option to hide player box</td>
<td>MEDIUM PRIORITY</td>
</tr>
<tr>
<td>Indication of scored points and lifes left</td>
<td>MEDIUM PRIORITY</td>
</tr>
<tr>
<td>Implement a highscore table</td>
<td>LOW PRIORITY</td>
</tr>
<tr>
<td>Better indication when player gets hit by an obstacle</td>
<td>LOW PRIORITY</td>
</tr>
<tr>
<td>Improve the impression of 3D</td>
<td>LOW PRIORITY</td>
</tr>
</tbody>
</table>

**Table 5.1: Requirements for prototype II**

Depending on the priority and the time, these requirements are going to be included in the next prototype. Also some minor issues, which are noted on the checklist, are going to be fixed.

**5.4 The second prototype**

The second prototype and final version is improved and refined from the first one. The functions and the handling are the same but the player can now choose a difficulty level (“easy”, “medium” and “hard”). The generation rate of the obstacles increases with the difficulty. Also the obstacles have now different shapes.

The player can now see his score and the lifes he has left.

The high priority tasks from table 5.1 have also been completed.

The access violation error, which occurred from time to time was a problem with the update function for the player box size. It could be fixed by setting an attribute of the player box. The tracking problems got fixed with a new setup for the cameras. They are now more focused on the feet and the head. It now works almost perfect. Also some adjustments (light intensity threshold) for the tracking server made some improvements.

Additionally, a new play mode have been invented where the player has to “catch” all objects. So it is the other way around as in the normal mode. This mode was a idea in the last minute to see, if people would enjoy that as well. It is very basic and has only one difficulty. Because of the time pressure, no more effort has been spent on further developments for this.

Basically, all requirements from table 5.1 could have been realized, except for the highscore table and adjustments for the 3D effects.
5.4.1 Screen shots

Here are some screen shots of prototype II. The menu bar has changed a bit. There are now more buttons to select the difficulty and the “catch” mode. The screen flashes red now, when the player gets hit by a box or if he misses a box in “catch” mode. Also the game over animation changed a bit and the player can see his score at the end. Furthermore, the shadow of the player box has been removed. This was not helping and actually more disturbing.

![Initial view with idle player box](image1)

*Figure 5.18: Initial view with idle player box*

![Screen shot while playing in normal mode](image2)

*Figure 5.19: Screen shot while playing in normal mode*

![Screen shot while playing in normal mode](image3)

*Figure 5.20: Screen shot while playing in normal mode*

![Screen shot while playing in “catch” mode](image4)

*Figure 5.21: Screen shot while playing in “catch” mode*
5.4.2 Evaluation

This is the result of the evaluation of the second prototype. As for the first prototype, 6 participants are asked to test the system. Each one tried the game modes “easy” and “catch”, with and without the player box. This was the basic test procedure. The ones, who had some energy left, tried “medium” and “hard” as well.

The questionnaire, that was used for the second prototype, can also be found in the appendix A.1. Most questions are the same (maybe rephrased a bit) as for prototype I. This allows a comparison to see, if the system got better or not.

Again, the following diagrams and descriptions are showing the result of the second prototype.
5. Results and Evaluation

5.4. The Second Prototype

**Figure 5.26:** Q.3 How convincing was the impression of 3D?

**Figure 5.27:** Q.4 Did you have the feeling of “being in the scene”?

**Figure 5.28:** Q.5 Which mode did you enjoy more?

**Figure 5.29:** Q.6 In normal (avoid) mode, which difficulty did you enjoy most?

**Figure 5.30:** Q.7 Do you prefer to see your player representation (the box) or not?
5. Results and Evaluation

5.4. The Second Prototype

Q.8 With the player-box, did you have problems with:

- orienting in the scene
- positioning yourself in the scene
- estimating distances
- estimating speed
- moving around

Figure 5.31

Q.9 Without the player-box, did you have problems with:

Figure 5.32

Q.10 If applicable describe the problems from question 8 and 9:

Most people had problems with estimating the distances with activated player box. Also some comments were made, that without the player box the game felt more real. With the player box it was some kind of “artificial” and the connection between the player and the box was not really there. This connection is not necessary, when the player box is not visible. The player has then more the feeling of being in the scene because it is a first person view.

Q.11 Did you have other problems? If so, please specify:

The attachment of the markers to the feet is still not perfect. Due to the fact, that the player is walking on the strap, the marker fell off sometimes. But this only happened to some testers.

Again, the tracking got lost some times. But it was definitely not that often as for
the first prototype. It seems, that the cameras are pretty much influenced by the
environmental light.
Also most people had troubles with the “catch” mode. It was too fast and the obsta-
cles were too small. This mode without the player box was pretty hard.

Q.12 Do you have ideas / suggestions for improvements and features?

Some participants asked for a more detailed player representation where the hands
are tracked as well. This would make the “catch” mode definitely more interesting.
Also separate legs would help for a better movement. This would be definitely
more natural, than calculating the centre of both legs to define the position of the
player box. Also different modes and speeds would make the “catch” mode more
interesting.
Most people mentioned the highscore again. Actually, this is pretty important for
a game. But because of the high time pressure, the implementation could not been
realized.
Other ideas are dynamic obstacles, which are changing the size and position while
they are moving towards you and a better indication where the player collided with
an obstacle.

Q.13 Do you have any other comments?

All participants had good fun with this game. Some people wrote, it is a pretty good
workout and might be perfect for a fitness program.

5.4.3 Summary

In direct comparison, the second prototype is a big improvement to the first one. The second
prototype didn’t crash once. It is very important to have a stable program. Also the tracking
could have been improved to establish a (almost) stable motion capturing.

Furthermore, prototype II is much more interesting with the three difficulty levels and the more
sophisticated obstacles. It was more demanding for the players.

The fact, that almost everyone preferred the view without the player box, is pretty interesting.
Actually, the game was designed with player representation, as third person view. To hide the
player box was just an idea to see, how the participants would do without the help of the box.
The comparison of figure 5.31 and figure 5.32 is pretty distinct. Especially the question for
estimating distances shows a definite result. Almost everybody did better without the player
box. Also the other questions in figure 5.31 and 5.32 are showing an increase in performance.
A closer look to the other questions (figure 5.25 and 5.27) with the differentiation between player box visible and player box not visible are indicating an improvement. The participants got faster a better feeling for the game.

Almost all requirements (see table 5.1) for the second prototype could have been implemented. This was very important in the development process, to get as much as ideas from the participants of the first prototype into the second prototype. Only the highscore table was too much work in the end as well as the improvements for the 3D effects.

The new game mode “catch” wasn’t that successful (see figure 5.28). That was probably, because the speed was pretty high and just too much for the player. Also the fact, that you can not catch the objects with the hands, maybe led to this result. Some improvements for this mode would probably increase the fun.

### 5.5 Evaluation results

This section summarizes the basic results of the two evaluation phases. First of all, the program evolved over time in a very positive way. A lot of problems could be fixed for the second prototype. Also a lot of improvements could be made with help of the first evaluation to increase the experience.

The implementation of different levels of difficulty made this game more accessible to a wider field of people. Now every one can choose, depending on the condition, a level and can increase the difficulty while progressing.

Also now, that the player can see the result at the end makes this game more game-like. The only negative fact is, that the highscore could not have been implemented.

Still a problem is the size of the lab. The space is very limited to create a perfect setup for the cameras. The area, in which the tracking is working fine, is quite small. Probably the tracking would be more stable with more cameras. Then the alignment can be refined to cover a greater area.

But in the central area, the tracking is working pretty fine. Also while jumping and ducking, the tracking doesn’t get lost or only for a very short time which doesn’t affect the movements.

Also the shadow of the player on the screen is pretty obstructive. This can not be changed because the system installed in the lab is a front projection. Maybe the screen could be moved a bit up. Then the projectors have to be calibrated again.

What the evaluation also revealed is, that a lot more improvements can be made to the program. This functionalities, which are implemented at the moment, are just the basics.

In terms of a game, there are a lot more functions, which could be implemented to make it more interesting and exiting.
What was marked as critical in the literature review was the latency of the tracking system. The fact, that the data has to be transmitted via the network, was quite risky because this might cause lags in the movement of the virtual environment.

Already the first prototype showed, that this was no problem at all and the response was quite reasonable.

Also the components of this system are pretty good working together. The C++ backend as a base and OSG as the frontend for the visuals. Also the tracking client could be implemented without problems. The collision detection system of OSG works fine and doesn’t need to be exchanged by another algorithm.
Final conclusion and future work

The last chapter of this document will summarize the project and explain what went well and what didn’t turn out that good. Also some ideas for future developments, improvements and extensions are described. The document will close with a personal conclusion.

6.1 Summary

The idea of the project “Ducking and Diving” was to realize a game, where the player has to move actively in front of a screen to avoid colliding with obstacles. The main goal of this project was to merge the real world with a virtual environment. So the player could interact with the VE. Also, since the gaming industry is going more and more to active sports, where the player has to be active to control the game, this project applies in this area as well.

Crucial for this project was to show, that such an interaction with a virtual world can be applied, especially in terms of real time applications. This project was analysing the bases of this problem.

The project started with a research for similar projects and helpful informations. The first chapter in this document is subdividing the problem into separate elements to get an overview of equipment and techniques, that are necessary for this work. It revealed, that the collision detection system is going to be a big part of this project, which needed a lot of precision work and tests. Also considerations were made about, which collision detection system to use. OpenSceneGraph has an own one but there are a lot of other frameworks and techniques.
As visual component, OpenSceneGraph is going to be used. There are some other graphic engines that could have been used but OSG seems to be the best option in terms of usability and handling.

Also the real time tracking system plays a crucial role. Since the game is not working without a proper tracking, it has to be adjusted very well. Also a head tracking system is used for this project. The head tracking is necessary to give the player the illusion of walking in the scene. The literature review describes some different approaches but the usage of the motion tracking system seems to be most applicable.

Furthermore, some investigation about the projection system has been done. The best option would be a back projection system to avoid shadows but in the lab is already a front projection system installed.

The chapter ends with a list of tasks and how difficult they are to realize.

After showing, what components the project requires and which techniques could be used, the project went on with a conceptional and requirement phase. This is chapter two in this document. It explains the tracking system, that has been used and how to connect the tracking server with the game.

Also the placement of the markers is described in order to get a stable tracking (that no marker is covered while playing). Furthermore, the connection of the markers to the virtual environment is explained (how the player is connect with the virtual player box).

Important is also the arrangement of the cameras. Some considerations have been made about this problem because the tracking has to be stable and not to interrupt the game play. Some different setups were tested in order to find the best solution. Also, the space in the lab is pretty limited, which is hindering the tracking.

The last part of this chapter describes the control program which was developed. Basically it has to bring the data from the tracking system to the visual component OSG.

Chapter four (the implementation) explains, how the control program was created. The introduction describes what programs and tools have been used in order to build the software.

After that, a description of each class follows, started with the main class. This class is the heart of the program and the entrance point of the software, which initializes all modules and starts the calculations. Also the loop, which create the frames for the visual output is located here.

Another big module is the tracking client. This component connects to the tracking server and requests the data. The server returns the locations of all markers. Now this module can calculate the new position and size of the player box. Also the head tracking is realized with this client.

The other classes, to provide the functions of the game, are Obstacle (representation of an obstacle with its functions), Player (representation of the player with its functions), Menu (the menu bar and the belonging functions) and some helper classes for debug purposes.
Furthermore, the scene graph of this application is shown. The scene graph describes the objects of the scene as a tree.
At the end is a description of the obstacles and its implementation and handling.

The next chapter describes the results and the two evaluations. After an introduction, explaining the development process, some instructions were given how to use the system.
After that, the first prototype was explained. First, some screen shots illustrate, how the visuals look like. On the next pages are the results of the evaluation phase, where the system has been tested by some participants. The main problem with this prototype were the weird access violations, which occurred from time to time and the tracking system.
These problems could be fixed for the second prototype. Also some new elements have been implemented (such as new obstacles, difficulty levels, new game mode, ...). Again, the section of the second prototype starts with showing some screen shots. On first sight, only the menu changed. But also a lot of details have been improved, like collision indication, colour / transparency of the player box and much more.
The evaluation of the second prototype indicated an improvement compared to the first one. This was proved by looking at the diagrams of both evaluations. It also turned out, that the game was more enjoyable without the player box. In this case, only the head tracking indicates where the player is standing right now. Most people found this much more immersive.
The end of this chapter explains the results of the evaluations.

In summary, this project can be seen as successful. It connected the real with the virtual world pretty well by creating an immersive game. The head tracking was used to let the virtual environment float which gives the illusion of walking in the real world.
The immersion could also be shown by some reactions from participants. For example, some of them tried to make them self as small as possible, when a narrow obstacle was coming.
So all of the project requirements could be fulfilled. In order to create a more exiting and thrilling game, a lot of work has to be done.
The next section will present some ideas for future developments.

6.2 Future developments and ideas
The game is in its current state completely functional and can be started immediately. But there are a lot of functions and options which are not included yet to make it to a even more exciting game.
The evaluations revealed, that this current state is just the very basics and that there is a lot of potential for improvements and extensions.

Some of the most important ideas are:

- **Highscore table**
  A table at the end, where the player can enter the name and see the ranking. This would require a file writer / reader in the code to store the data.

- **A more detailed avatar**
  At the moment, the player is represented by a box. This is very abstract. Also the movement is pretty unnatural. Better would be to track the feet separately. Also interesting would be to track the hands, especially for the catch mode. This would improve the handling enormously.

- **Dynamic obstacles**
  Dynamic obstacles are changing the size and the position while they are moving towards the player. This would increase the level of difficulty and give the game more thrill. For that, the class `Obstacle` has to be changed completely. Also some round obstacles should be planned.

- **More / different modes for catching**
  The catch mode, that was implemented in the second prototype, was just a test to see, how people would like it. It has potential but as well as for the normal mode, different levels of difficulty have to be developed.

- **Dynamic difficulty increase** At the moment, the player can choose one difficulty level which stays the same till the game is over. Better would be, if the difficulty increases while playing. For example, if the player passed ten obstacles, the difficulty changes to a higher level.

- **Better indication, where the collision happened**
  To increase the performance of a player, he has to see if he is doing good or bad. A better indication, where the collision happened, would help the player to see what he has done wrong.

- **Multiplayer mode**
  A multi player mode would be very interesting for this game. Two players for example, could play on one stage at the same time. Also a split screen would be possible. The one with the most lifes at the end wins. For that, a separate player representation has to be implemented as well as a second counter for the collisions.

- **More colours**
  The environment is pretty sterile at its stage right now. Everything is kind of grey and
doesn’t look very attractive. A game of that kind should have some fancy designs and animations so that the player is not bored after a while because of the boring colours.

- **More statistics**
  The games only shows the scored points at the end of the game. But also interesting would be some more statistics like: highest jump, longest time without a collision, walked distance, . . .

- **Improve the attachment of the markers to the feet**
  The attachment of the markers to the feet is not perfect. When walking around, the rubber strap got lost sometimes from the foot.

- **Self-calibration**
  A big problem is that when the cameras get changed, the system has to be calibrated again. This can only be done by setting parameters in the code. The software has to be compiled again. Better would be a configuration file, an input dialogue or a self calibration function, which only has to be run once at the beginning of the game.

- **Player can control the game**
  At the moment, the only option to start the game, is with the computer mouse. This is not very good when the player is standing in front of the screen. An option would be to implement a Wii Remote. Another solution could be, that the player has to walk to a specific area on the field to start the game. Also gesture recognition could be implemented to control the game.

- **Pause button**
  Every game has a pause button. This function is still missing in this game. It could be used, when the player lost a marker.

### 6.3 Personal conclusion

I absolutely enjoyed working on this project. It was a great opportunity for me, to work with such an equipment.
I also liked the practical tasks, meaning programming and setting up the tracking system. A lot of facts could have been discovered by “trial and error”.
Also the fact, that this application is a combination of sport and video games, was very interesting for me. The game is pretty addictive and exhausting at the same time, at least in my case.

Also, this project gave me the opportunity to refine my skills in C++ programming. Since I have a strong Java background, I had some problems getting into the philosophy of pointers again.
This led to access violation errors and I was struggling a lot. But in the end, I managed to get rid of them.

A big problem was the time management. I wanted to create a pretty good and precise working system. But this took me longer than expected and I had to drop some enhancements. Three month are just not enough for such a project and to write a report about it. It really hurts me to end my project like this without polishing and refining. Because of the huge time pressure, I had to rush though some modules and couldn’t test them extensively. That is also why the first prototype was quite unstable. But the second prototype and final version works pretty reliable.

In the end I can say, that this project was very exciting and I learned a lot about virtual environments and motion capturing. Also the evaluation phases showed me, that tests by other people are very important and could help to improve the performance of an application a lot.

With this project as a base and my education at the Heriot Watt University, I see myself pretty good prepared for a job in this area.
Bibliography

All references to online resources have last been checked on August 19, 2010.


List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>The system components and its difficulty</td>
<td>14</td>
</tr>
<tr>
<td>4.1</td>
<td>Keys and its actions</td>
<td>41</td>
</tr>
<tr>
<td>5.1</td>
<td>Requirements for prototype II</td>
<td>54</td>
</tr>
</tbody>
</table>
## List of Figures

1.1 Screenshot from the project "Virtual Tübingen" [webh] . . . . . . . . . . . . 1  
1.2 Mercedes designers in a cave [weba] . . . . . . . . . . . . . . . . . . . . . . 2  
2.1 The basic components of the system . . . . . . . . . . . . . . . . . . . . . 6  
2.2 Example screen for the system . . . . . . . . . . . . . . . . . . . . . . . . 7  
2.3 Different stages of a collision . . . . . . . . . . . . . . . . . . . . . . . . . 8  
2.4 How motion capturing works [webk] . . . . . . . . . . . . . . . . . . . . . 10  
2.5 Example for head tracking glasses [SR05] . . . . . . . . . . . . . . . . . . 12  
2.6 Johnny Lee . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12  
2.7 The passive stereo system chart . . . . . . . . . . . . . . . . . . . . . . . . 13  
3.1 Motion capturing cameras . . . . . . . . . . . . . . . . . . . . . . . . . . . 17  
3.2 Marker . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17  
3.3 Screenshot of the Qualisys Track Manager . . . . . . . . . . . . . . . . . . 18  
3.4 Representing the player (with markers) as a box . . . . . . . . . . . . . . . . 19  
3.5 Photo of the camera setup . . . . . . . . . . . . . . . . . . . . . . . . . . . 20  
3.6 Christie LX45 projector . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21  
3.7 Motion capturing server and the client . . . . . . . . . . . . . . . . . . . . . 22  
3.8 Flow chart of the test client . . . . . . . . . . . . . . . . . . . . . . . . . . . 22  
3.9 Example screen with descriptions . . . . . . . . . . . . . . . . . . . . . . . 23  
4.1 The first few constants of the CONSTANTS.h . . . . . . . . . . . . . . . . . . 26  
4.2 Attributes and function of the main . . . . . . . . . . . . . . . . . . . . . . 27  
4.3 Flow chart of the main . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 28  
4.4 Tasks in the main drawing loop . . . . . . . . . . . . . . . . . . . . . . . . . 29  
4.5 Functions of the RTCIent . . . . . . . . . . . . . . . . . . . . . . . . . . . . 30  
4.6 The flow chart of the RTCIent . . . . . . . . . . . . . . . . . . . . . . . . . . 32  
4.7 Class diagram of OpenSceneGraph related classes . . . . . . . . . . . . . . . 34  
4.8 Attributes and functions of class Obstacle . . . . . . . . . . . . . . . . . . . 35  
4.9 Attributes and function of class Player . . . . . . . . . . . . . . . . . . . . . 37  
4.10 Attributes and functions of class Menu . . . . . . . . . . . . . . . . . . . . . 38
4.11 Attributes and functions of class KeyboardEventHandler
4.12 Functions of the class Utilities
4.13 The objects in the scene graph
4.14 The implemented obstacles
5.1 The development process
5.2 The markers for the feet and the prepared glasses
5.3 A player equipped with markers on the feet and the head
5.4 Screen shot of the Qualisys tracking manager with loaded model
5.5 Initial view
5.6 The player box with the start sequence
5.7 Screen shot while playing
5.8 Screen shot while playing (player is ducking)
5.9 Player got hit by an obstacle
5.10 Game over animation
5.11 Q.1 How difficult was it to use the program in general?
5.12 Q.2 Do you think the movement of the player-box matched your movement
5.13 Q.3 How long did it take you to get the right feeling for the distances to the obstacles?
5.14 Q.4 How convincing was the impression of 3D?
5.15 Q.5 Did you have the feeling of “being in the scene”?
5.16 Q.6 What would you say about the speed of the obstacles?
5.17 Q.7 Did you have problems with:
5.18 Initial view with idle player box
5.19 Screen shot while playing in normal mode
5.20 Screen shot while playing in normal mode
5.21 Screen shot while playing in “catch” mode
5.22 Screen turns red when player got hit by an obstacle
5.23 Game over animation and results
5.24 Q.1 When the player-box is visible, do you think the movement of the player-box matched your movement
5.25 Q.2 How long did it take you to get the right feeling for the distances to the obstacles?
5.26 Q.3 How convincing was the impression of 3D?
5.27 Q.4 Did you have the feeling of “being in the scene”?
5.28 Q.5 Which mode did you enjoy more?
5.29 Q.6 In normal (avoid) mode, which difficulty did you enjoy most?
5.30 Q.7 Do you prefer to see your player representation (the box) or not?
5.31 Q.8 With the player-box, did you have problems with:
5.32 Q.9 Without the player-box, did you have problems with:

VII
Code Listings

4.1 XML stream fragment from the server .......................... 33
Acronyms

**3D** 3 dimensions

**AIM** Automated Identification of Markers

**API** Application Programming Interface

**DOF** Degree of freedom

**DVI** Digital Visual Interface

**fps** frames per second

**IDE** Integrated Development Environment

**IP** Internet Protocol

**IR** infra red

**OSG** OpenSceneGraph

**QTM** Qualisys Track Manager

**TCP** Transmission Control Protocol

**VE** Virtual Environment

**XML** Extensible Markup Language
Appendix

A.1 Feedback sheets

On the following pages are the feedback sheets, that have been used for the participants. Both (prototype I and II) are basically the same, only that the second one had been extended because of the improvements and enhancements.
A.1.1 Questionnaire prototype I

Ducking and Diving - Prototype I

Questionnaire # ______

Q.1 How difficult was it to use the program in general?

<table>
<thead>
<tr>
<th>very hard</th>
<th>hard</th>
<th>neutral</th>
<th>easy</th>
<th>very easy</th>
<th>N/A</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Q.2 Do you think the movement of the player-box matched your movement ...

<table>
<thead>
<tr>
<th>not at all</th>
<th>sometimes</th>
<th>ok</th>
<th>pretty good</th>
<th>perfect</th>
<th>N/A</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Q.3 How long did it take you to get the right feeling for the distances to the obstacles?

<table>
<thead>
<tr>
<th>very long</th>
<th>long</th>
<th>neutral</th>
<th>short</th>
<th>very short</th>
<th>N/A</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Q.4 How convincing was the impression of 3D?

<table>
<thead>
<tr>
<th>not at all</th>
<th>a bit</th>
<th>good</th>
<th>almost perfect</th>
<th>perfect</th>
<th>N/A</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Q.5 Did you have the feeling of “being in the scene”?

<table>
<thead>
<tr>
<th>not at all</th>
<th>a bit</th>
<th>good</th>
<th>almost perfect</th>
<th>perfect</th>
<th>N/A</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Q.6 What would you say about the speed of the obstacles?

<table>
<thead>
<tr>
<th>way too fast</th>
<th>too fast</th>
<th>ok</th>
<th>perfect</th>
<th>too slow</th>
<th>way too slow</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Q.7 Did you have problems with:

<table>
<thead>
<tr>
<th>orienting in the scene</th>
<th>serious problems</th>
<th>some problems</th>
<th>neutral</th>
<th>no problems</th>
<th>N/A</th>
</tr>
</thead>
<tbody>
<tr>
<td>positioning yourself in the scene</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>estimating distances</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>estimating speed</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>moving around</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Q.8 If applicable describe the problems from question 7:

______________________________________________________________________________________
______________________________________________________________________________________
______________________________________________________________________________________
______________________________________________________________________________________
______________________________________________________________________________________

- Please Continue -
APPENDIX

A.1. FEEDBACK SHEETS

Q.9 Did you have other problems? If so, please specify:
______________________________________________________________________________________
______________________________________________________________________________________
______________________________________________________________________________________
______________________________________________________________________________________
______________________________________________________________________________________

Q.10 Do you have ideas / suggestions for improvements and features?
______________________________________________________________________________________
______________________________________________________________________________________
______________________________________________________________________________________
______________________________________________________________________________________

Q.11 Do you have any other comments?
______________________________________________________________________________________
______________________________________________________________________________________
______________________________________________________________________________________
______________________________________________________________________________________
A.1.2 Questionnaire prototype II

Ducking and Diving - Prototype II

Questionnaire # ________

Q.1 When the player-box is visible, do you think the movement of the player-box matched your movement …

<table>
<thead>
<tr>
<th>not at all</th>
<th>sometimes</th>
<th>ok</th>
<th>pretty good</th>
<th>perfect</th>
<th>N/A</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Q.2 How long did it take you to get the right feeling for the distances to the obstacles?

<table>
<thead>
<tr>
<th>with player-box</th>
<th>very long</th>
<th>long</th>
<th>neutral</th>
<th>short</th>
<th>very short</th>
<th>N/A</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>no player-box</th>
<th>very long</th>
<th>long</th>
<th>neutral</th>
<th>short</th>
<th>very short</th>
<th>N/A</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Q.3 How convincing was the impression of 3D?

<table>
<thead>
<tr>
<th>not at all</th>
<th>a bit</th>
<th>good</th>
<th>almost perfect</th>
<th>perfect</th>
<th>N/A</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Q.4 Did you have the feeling of "being in the scene"?

<table>
<thead>
<tr>
<th>not at all</th>
<th>a bit</th>
<th>good</th>
<th>almost perfect</th>
<th>perfect</th>
<th>N/A</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>with player-box</th>
<th>both</th>
<th>catch mode</th>
<th>N/A</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>no player-box</th>
<th>both</th>
<th>catch mode</th>
<th>N/A</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Q.5 Which mode did you enjoy more?

Q.6 In normal (avoid) mode, which difficulty did you enjoy most?

<table>
<thead>
<tr>
<th>easy</th>
<th>medium</th>
<th>hard</th>
<th>N/A</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Q.7 Do you prefer to see your player representation (the box) or not?

<table>
<thead>
<tr>
<th>yes</th>
<th>doesn't matter</th>
<th>no</th>
<th>N/A</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Q.8 With the player-box, did you have problems with:

<table>
<thead>
<tr>
<th>orienting in the scene</th>
<th>serious problems</th>
<th>some problems</th>
<th>neutral</th>
<th>no problems</th>
<th>N/A</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>positioning yourself in the scene</th>
<th>serious problems</th>
<th>some problems</th>
<th>neutral</th>
<th>no problems</th>
<th>N/A</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>estimating distances</th>
<th>serious problems</th>
<th>some problems</th>
<th>neutral</th>
<th>no problems</th>
<th>N/A</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>estimating speed</th>
<th>serious problems</th>
<th>some problems</th>
<th>neutral</th>
<th>no problems</th>
<th>N/A</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>moving around</th>
<th>serious problems</th>
<th>some problems</th>
<th>neutral</th>
<th>no problems</th>
<th>N/A</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Please Continue -

XIII
Q.9 Without the player-box, did you have problems with:

<table>
<thead>
<tr>
<th>Activity</th>
<th>Serious Problems</th>
<th>Some Problems</th>
<th>Neutral</th>
<th>No Problems</th>
<th>N/A</th>
</tr>
</thead>
<tbody>
<tr>
<td>orienting in the scene</td>
<td>Q</td>
<td>Q</td>
<td>Q</td>
<td>Q</td>
<td>Q</td>
</tr>
<tr>
<td>positioning yourself in the scene</td>
<td>Q</td>
<td>Q</td>
<td>Q</td>
<td>Q</td>
<td>Q</td>
</tr>
<tr>
<td>estimating distances</td>
<td>Q</td>
<td>Q</td>
<td>Q</td>
<td>Q</td>
<td>Q</td>
</tr>
<tr>
<td>estimating speed</td>
<td>Q</td>
<td>Q</td>
<td>Q</td>
<td>Q</td>
<td>Q</td>
</tr>
<tr>
<td>moving around</td>
<td>Q</td>
<td>Q</td>
<td>Q</td>
<td>Q</td>
<td>Q</td>
</tr>
</tbody>
</table>

Q.10 If applicable describe the problems from question 8 and 9:

______________________________________________________________________________________
______________________________________________________________________________________
______________________________________________________________________________________
______________________________________________________________________________________
______________________________________________________________________________________

Q.11 Did you have other problems? If so, please specify:

______________________________________________________________________________________
______________________________________________________________________________________
______________________________________________________________________________________
______________________________________________________________________________________
______________________________________________________________________________________

Q.12 Do you have ideas / suggestions for improvements and features?

______________________________________________________________________________________
______________________________________________________________________________________
______________________________________________________________________________________
______________________________________________________________________________________
______________________________________________________________________________________

Q.13 Do you have any other comments?

______________________________________________________________________________________
______________________________________________________________________________________
______________________________________________________________________________________
______________________________________________________________________________________
______________________________________________________________________________________

______________________________________________________________________________________