
# Augmented Reality Mario Kart Race Game

## Honours Project Final Report

Yosemite Sam

2005xxxxx

BHGS_4

Project Supervisor: Jonathan Sykes

Second Marker: Richard Foley

"Except where explicitly stated all work in this document is my own"

Signed:_____Date:__/__/__

# Abstract

This report is an accompanying document to a program created to investigate the use of augmented reality in the games industry. In this project before anything was started coding wise a literature review was carried out in order to find out about the subject area. A look into what augmented reality actually is was first on the agenda. After finding what it is, it was decided to look at how it has been used in the past. From this the research question was taken. So the aim for this develop and test project is to create a computer game using techniques of augmented reality normally found in different industrial settings.

With the focus of research found it was then down to designing what functionality should be incorporated into the game. The main feature of the game was using augmented reality. This would be done using a robot with an inbuilt webcam which would relay the pictures to a computer. The game would then overlay 3D models onto the pictures from the camera. These models would then be interactive to the player. The player would be able to pickup some objects and shoot other objects.

When the implementation started the world was set up to allow objects to be added into it. When the objects were added into the world it was found they couldn't be positioned directly. This took a while figure out why, but was eventually found. This was one of many set backs during the implementation and due to this the robot was not used in the game. Instead a robot a textured background was used. This was created using a 360 degree picture, and worked well.

During the implementation testing was being carried out with each new piece of code written. Each feature was checked before a new one was implemented. This meant the project was an iterative process.

When the project coding was complete it was then evaluated in house by the development team. In order to evaluate the program it was decided to check the functionality compared to the functions of the industrial uses of augmented reality. A look at the functionality of Mario Kart was also checked as this was the style of game to be created in this project.

After evaluating the game conclusions were drawn from the findings, these conclusions were used to answer the research question of the project.

## Acknowledgements

I would like to thank my supervisor, Dr. Jonathan Sykes, as without his idea in the first place this project would not have been able to be done. I would also like to thank my second supervisor, Peter Barrie, his technical knowledge was very useful to the completion of this project.

I would like to thank my family for being supportive during the run of the project. I'd like to thank my work for allowing me to have time off to concentrate on the project.

# Table of Contents

# 1.0 Introduction.

To begin this report there will be a summary of what this project is about. It will include why the project is taking place as well as how it will be carried out.

## *1.1 Background.*

This project is about the use of a technology known as Augmented Reality (AR) to create a computer game in the style of Mario Kart. The game will use a robot which has a webcam attached to the front of it, sending the images to a computer the player using. The images will then be processed and depending on pixel colours, different 3D models will be overlaid onto the screen. The robot being used is the Surveyor SRV1.

AR techniques used within other industries can be looked into and some ideas can be taken from these. Industries using augmented reality include; medicine (Haugh 2005), motor industry (Anastassova, Burkhardt 2007) and more importantly the entertainment industry (Pretlove 1998) including movies and games.

However in uses of AR in games there has not been anything which involves a moving camera only moving objects in the cameras field of view. This would suggest there is a need for this project.

### 1.1.1 The Problem

Augmented Reality has been used in the games industry in the past, in games such as Eye of Judgement (Anon 2008a) on the Sony Playstation 3 and also the Eyetoy Play (Andrews 2005) series on the Playstation 2. So AR is proven to work in real-time game environments. The games mentioned above do not actually have any movement of the camera being used so do not pose the problem of depth for the models appearing on the screen. Therefore a look into the other uses of AR will be needed.

AR is used in various different environments from operating theatres (Haugh 2005) to car garages (Anastassova, Burkhardt 2007), so methods could be used from these different areas within this project. A good example of what this project would like to do is, a look at the heads-up-displays (Anon 2004) found in fighter jets, which track other jets with boxes around them. Although this project is creating a game it is a similar kind of look that is desired except the boxes being changed to 3D models. The technology used to locate the positioning for the objects is very different but this example gives a good visualisation of the type of thing this project is trying to achieve.

The controls for the robot are also a very important part of this project, as it is one of the main parts of the project. A look at the different methods of robot controls will need to be looked into. Robots can be either autonomous or remotely controlled . As this robot is going to be used in a game it is necessary for the robot to be remotely controlled by the user.

As the robot is going to be manipulated through a computer there will need to be a

network connection to the actual robot. It could be done either wired or wireless, it would be desirable to use wireless technology as it would give the robot better freedom rather than trailing a wire around with it.

## *1.2 Project Outline*

This project will involve creating a piece of software which controls a robot which is on a wireless network. The robot being used for the development will have an integrated webcam onboard. The controlling program will use the images sent back from the webcam and add computer generated 3D models into the real world image. This is where the augmented reality is brought into play; the 3D models will be items such as targets and missiles.

In order to create the 3D images Softimage XSI will be used. The models will be displayed within the program by using an image processing (Day, Knudsen ) technique. This will involve checking each of the pixels on the images coming from the camera, 3D images will be displayed if a pixel is a certain colour. A tricky part of this would be checking for the distance in terms of 3D space as the images being processed is 2D.

Another method thought about for placing objects would be creating a 3D space which the user cannot see, for example all the user would need to do is make sure there is a big enough space in the real world that the virtual world can fit into. Using this method would mean that the objects would be in a predefined place every time you play, unlike the image processing technique where you would need to add objects in the real world to make objects appear, however the positioning of objects could be different each play

through.

### 1.2.1 Research Question

The aim of this project is to answer this question:

Can the techniques used out with the games industry augmented reality systems be used to create a computer game which controls real life robots in a racing style game?

In order to help answer this question a set of objectives will need to be created to make sure the literature review stays on track with the project. These objectives are as follows:

1. Find out what augmented reality is.

This is one of the most important points to the project as if it is not understood what augmented reality is then the research question cannot be answered.

2. Find out where augmented reality has been used in the past.

This involves finding out how the technology has been used previously. As the project is trying to use Augmented Reality techniques used in industrial settings to create a game, the way it is used in industry is essential to the development.

3. Find out how image processing is used with Java.

As the project is going to be making use of image processing it would be beneficial to check how it can be used within the platform being used, which will be Java in this case.

4. Find out how to gain control of robots.

In order to make sure the robot will be able to be manipulated easily it would be useful to find out how to manipulate robots. This should help make sure the robot is being controlled efficiently.

5. Find out how the robot has been used in the past

Knowing about how the robot has been used in the past can be very helpful as it can give ideas as to how it could be used. It could also help if problems occur as someone may have experienced the same problem before.

6. Find out how augmented reality has been used in games.

To find out if it is feasible to create a game using augmented reality it would be a good idea to find out if and how it has been used in games in the past.

With the objectives of the literature review given previously there is an initial hypothesis to take into account, this hypothesis us:

It will be possible to create a game using Augmented Reality methods used in

industry.

This hypothesis has been used as it would seem that the methods used within industry would fit quite well into a game setting. So from this hypothesis it is hoped that the project will create a successful game which uses the augmented reality technology, which has been influenced by industrial uses.

# 2.0 Literature Review

In this section the objectives from the previous section will be looked into with the use of reading some literature. This section will try to give an answer to each of these objectives.

## *2.1 What is Augmented Reality?*

Augmented reality is a technology which can be used to alter the real world. It can do this in various different ways, (Anon 2004) states that Heads Up Displays (HUD) are being used in cars to give the driver information such as speed. This HUD is projected onto the windscreen, in the BMW 5 series, therefore giving the driver details without him/her having to take their eyes off the road. This is a very primitive form of augmented reality, and has been adapted from the use of HUDs in fighter jets, which are much more complex as they need to track other objects with radar and display boxes round these items for the pilot to see. The use of a HUD could potentially be a good idea for the game as it can be a very useful way of showing information to the player.

Augmented reality can also be seen in the movies (Pretlove 1998). With the use of special effects movies are enhancing the real life situations, with the use of green screen, which is the actors standing in front of a green background then in post production a new backdrop is added digitally. Computer Generated Images (CGI) are also used in the movie industry, this is the use of 3D models within a live action movie. In fantasy or science fiction movies where there are aliens or other creatures this can be

very helpful as people will not need to dress up in costumes. However the downside to this is the images are quite easy to tell from the real life images. This would not be as useful when creating a game as it is done all post production and not in real-time.

Augmented reality can also take the form of overlaying 3D models over 2D images of the real world in real time. For example (Anon 2008b) shows the use of augmented reality in this way, with a card being used in front a camera which recognises a special code on the card to create a floating video box which can be manipulated by moving the card. This is the type of augmented reality that this project would be using as a game is a real time situation.

This has been a helpful look into what augmented reality is, it shows that there isn't just one way to do it. This shows that there are different styles of augmented reality such as HUDs, post production special effects and real-time augmented reality. In essence augmented reality is manipulating real life images with computerised overlays.

Asking this question has been a vital part of this project. After looking over different literature, an understanding of the technology has been arrived at. With the different styles of augmented reality found it would be better to look into the real-time application use further as that is the most appropriate for this specific project.

## *2.2 What settings have augmented reality been used in?*

There was a brief look at how augmented reality has been used in the previous objective. In this part a deeper discussion will take place.

After finding out what augmented reality is, it was important to find where it has been used previously. One of the first uses of augmented reality found was it being used by car mechanics as a teaching aid for trainee mechanics (Anastassova, Burkhardt 2007). This article describes the use of augmented reality being used for training purposes. This augmented reality works with having a screen with a car on it and the car has all its different parts labelled with a box around them highlighting what they are and where they go.

This could be helpful when creating the game as it can be used in the same sort of way with having certain things recognised and labelled accordingly, such as where walls or targets should be placed. This method could also be used if other robots are incorporated into the game which could be in different teams and could be labelled so the player can differentiate between team mates and enemies.

Another use of augmented reality found was the use of it to help surgeons with operations (Haugh 2005). In this article augmented reality is used in keyhole surgery, where the surgeon has control of a machine which uses two screens to relay pictures from the endoscope inside the patient. One screen shows the real life images, and the other uses 3D models created from the use of x-rays to show a clearer picture of the inside of the patient. Because of the second view, with 3D images, the machine could serve as a training tool as well. This would allow a surgeon to practise an operation without putting anyone in danger.

This is a clever way to show how augmented reality works, having two separate

displays, one showing the real life view, not enhanced in anyway, and the other showing what it looks like with the augmented reality turned on. This could be used in the project to show the player what it looks like without the use of augmented reality, although it would mean that the user would not be able to play the game as it is all orientated around the augmented reality. This could be used but there would need to be two displays as there won't be any targets or walls to be seen if the augmented display is not used.

Another use of augmented reality is in heads up displays (HUD) (Anon 2004). These displays are making their way into cars, such as the BMW 5 Series and X5 range. The HUDs are quite basic giving the driver his speed and a rev-counter. This is helpful as the driver does not need to take his eyes off the road when he wants to find out his speed, making it safer. The HUDs found in cars are simplified versions of those found in fighter jets, which not only give the pilot information such as speed, but also track targets.

A HUD is a good idea for use in this project because it can give the player helpful information such as positioning, speed and weapon status. HUDs are featured in most games and are very helpful when playing, soothe use of HUD in this game will make it better for the user as it is something people are familiar with.

Another use of augmented reality is education, (Ashley 2008) talks about "magic books" which is children's books which have special barcodes within them which can be recognised by specialist software which can be put onto mobile phones. The example used by (Ashley 2008) is that of a book where an animal on a page comes to life when

viewed through a camera phone. The author of this article also says that this sort of technology could be used well within the marketing world. Pepsi has used a similar technology on the side of their cans, some Pepsi cans have a special barcode on the side of them and when looked at through a mobile phone it goes to the Pepsi website on the phone providing it has an internet browser.

The main idea which can be useful from this article is the use of image recognition of the barcodes in this article. The article does not go into the detail of how this is achieved but it shows what is possible.

There are many different industrial settings which use augmented reality. Despite the industries being very different the use of augmented reality is relatively similar. The different industries use augmented reality with overlays of 3D images. The different industries have different reasons for using augmented reality though. Ideas can be taken from each of the different industries, so it is has been good to find out how the technology has been used. Functions which could be used within the game are; the use of labelling objects which have been recognised, this could be used for debugging purposes. A two screen display could be useful when creating the game too, it could be used in debugging also where image recognition is used to make sure the correct models are being displayed at the right time, for example a target will only be overlaid when a certain colour is recognised. The heads up display should also be used within the game as it is a good source of information for the player. It will need to be decided what to add to the HUD as if there is too little on it then the user night not know enough, likewise if there is too much on it then the user may become confused. Finally it would be wise to use image recognition as it can make the game less linear than if it isn’t used.

This is because the game can be changed each play through by moving real life objects about. So a look at the past uses of augmented reality has been helpful for ideas of what to add into this project.

## *2.3 How does image processing work with Java?*

Image processing will be a key part to the operation of augmented reality in this project. As java will be used in during the development stage of this project it will be valuable to find out how to program for the use of image processing using java.

The javaworld website (Day, Knudsen ) was used to find out about how to use image processing for 2D java. The website has a good tutorial for image processing which will be put to good use in this project.

Image processing is changing the appearance of a digital image (Day, Knudsen ) which can then be used later for the purpose desired. In this project it will be used to create overlaid images on top of real life images, giving targets and missiles to create a game.

Image processing basically takes in each of the pixels in an image and manipulates them depending on what colour it is. (Day, Knudsen ) discusses three different techniques for image processing, these are; Convolution, Look-up Tables and Thresholding.

Convolution uses the pixels surrounding the one being processed to change the colour value of that pixel, using the percentage of colour of each other pixel. The percentage of the surrounding pixels is governed by a 3x3 matrix, and different effects can be created

from using different percentages (Day, Knudsen ). This is not useful for this project as this project is trying to add images into the overall image rather than just change the colours of the existing image. There is one matrix that can come in useful however as it can be used for edge detection, this could be useful because it could be used to add images in corners for example.

Look-up tables use the colour values of each of the red, green and blue to change the colour of the pixels. The main use for this is creating effects such as negatives or changing how much of each colour is used (Day, Knudsen ). It does this by inverting the colour values of each colour. Posterization is another use for this as the colour values can be reduced by percentages. Look up tables do not seem to have any use in this project as it is only used for creating new effects for the images.

Thresholding is basically manually setting what colours can be used. A maximum and minimum are set, then a threshold value is set and any colour which is above this threshold value will be set to the maximum and anything under the threshold will be set to the minimum (Day, Knudsen ). This doesn't really have much relevance to this project because it is basically just changing the effect on the image.

The most helpful thing taken from image processing would be the use of an edge detection method, because it could be used for putting overlays on top of the edges. Other techniques mentioned in this article could be used, for example if the player picks up power-ups in the game then it could trigger a screen effect such as blurring the screen or making the screen change to a certain colour to give the game extra features. Although this article was not quite giving what was desired, it did give some new ideas

which otherwise would not have been thought of.

## *2.4 How are robots controlled?*

In order to find out how robots can be manipulated, a look into the different methods for controlling robots was looked into. There were two main categories of robots found; they were; autonomous and remote controlled. In the article by (Anon 1994) it explains that robots have been getting more advanced over the years but the software used to control them has not been as rapidly evolving.

The article mentioned above also talks about the use of lasers to guide the robot, this is more oriented towards an autonomous robot. This project will not be using autonomous robots so this paper is not very useful.

Another article which dealt with controlling robots (Moallem, Khoshbin 2006) talked about the programming languages used in robots. This article stated that in the early days of robots, each robot would have its own unique language, this would have been very difficult to deal with as if a programmer would like to work with many different robots they would have to learn a new language for each robot they use. Now though robots can use languages such as C++ and java. This is good to know as the programmer can use skills they already have without having to learn a new language. (Moallem, Khoshbin 2006) also talks about how to program for robots, it recommends to keep the program as modular as possible as it can help for code reusability. This is a programming technique which should be incorporated into this project as it makes the program easier to read and understand, commenting should also be used so others can

understand what the program is doing. Another technique (Moallem, Khoshbin 2006) talks of is the use of a robot class which holds all the attributes for the robot and means new robots can be added very easily.

The last article looked into about robots was about the way they steer (Tlale 2006). This article talks about the use of two different techniques used to steer the robot, using different types of wheel layouts, the first layout was like a car, with four wheels and two of the four would be used to steer and drive, this is the sort of layout a front wheel drive (FWD) car has. The other type of wheels used were called “Macanum wheels” (Tlale 2006), these allow the robot to change direction without changing the way it is facing, meaning all the wheels on the robot can steer. Initially this seemed to be quite irrelevant to this project as the Surveyor SRV-1 robot being used in the experiment ha tracks rather than wheels but it has been useful as it gives a good idea of the way a robot can move. Thought needs to be put into how the tracks actually work, as the wheels within the tracks do not turn form side to side, so in order to turn the tracks need to be moving in opposite ways from each other or one track can move faster than the other.

This has been a valuable piece of research due to the realisation that robotics has been slow in being programmable. As mentioned above in the past robots previously had their own languages which were unique to each type of robot, but nowadays robots programmable through some major high level languages such as C++ and java. This has been a big relief to the programming team as they already had experience of these languages. Remote control robots were not really featured in these papers above, autonomous robots were talked about more, this did not really effect how well the research was as it was looking at how the robots are controlled physically rather than

how their AI worked.

## *2.5 How has the Surveyor SRV-1 been used in the past?*

The robot being used in this project is the Surveyor SRV-1 (Anon ). The SRV-1 is a small robot which is made up of a few different circuit boards. The circuit boards can be interchanged; there is a board which has a camera on it. This is the layout that will be used in this project. It is useful to know how this robot has been used in the past as there could have been something similar to what this project is trying to do, therefore some ideas could be taken from previous work.

In the online surveyor journal (Anon ) many people have added experiments they have taken part in. There have been many different experiments taken place using this robot. Some of the different experiments will be talked about in this section.

There has been a program available which is used to control the robot, this is using the java console (Anon ). This program allows a user to manipulate the robot by moving around, switching on the laser pointers mounted on the SRV-1. This program displays the pictures being streamed from the camera and displays them on the screen. This program could be used as a basis for the programming in this project. This program is useful to look at because it is in the same general area as what this project is trying to do.

Other experiments involve using image processing with the SRV-1. In one experiment the use of edge detection has been used (Anon ). This has been discussed before in the

image processing section. This is helpful as it shows that image recognition works with the robot being used.

There has been some interesting work done with the use of stereo vision (Anon ) using the SRV-1. This is done by using two cameras creating a two layered display on the screen, which when viewed with 3D glasses allows the user to view the illusion of a 3D image.

Although there has been work with image recognition using the SRV-1, there has not been any augmented reality applications been made featuring this robot. This means that this project is quite relevant because it has not been done before.

Reading through this journal has shown what is possible from using the robot. The most beneficial part of this journal is the program that it uses to manipulate the robot as it can be taken apart to look at what it actually does to make the robot work. The other major benefit was finding out that the robot can be used for image recognition techniques such as edge detection, which were looked into earlier and were decide were a good idea to use for placing objects on the screen.

## *2.6 How has augmented reality been used within games?*

In order to check if augmented reality has a place in the games industry a look into what is already on the market has been useful to gauge whether AR can work in games. There have been a few games using augmented reality in the past.

One of the first to uses of augmented reality in the games industry was for games using the Sony Eyetoy, these games are controlled through gestures of the player. The player stands in front of the Eyetoy camera and the pictures are displayed on the screen, depending on the game played a different overlay is put on the screen leaving a space for the player. The player would then move his/her arms around to interact with the overlay. The third game in the Eyetoy: Play series was found in a review in a newspaper (Andrews 2005) the review is brief but talks of how enjoyable the game is. This shows there is a market for games with augmented reality, it also shows that people have enjoyed this type of game for quite long as it is the third game in the series and is still found to be enjoyable.

Another more recent game to feature augmented reality is Eye of Judgement (Anon 2008a) also another product of Sony using the Sony camera for the Playstation 3. Eye of Judgement is a card game which brings the cards to life when put under the camera. The cards have special barcodes on them, which are recognised by the camera software and produce monsters on top of the cards on the screen. Different cards give different monsters and can fight against each other. This also shows a market for augmented reality in games, although the market which this game appeals to may be both card game players and computer games players.

Another magazine article (Anon 2008b) talks of an exhibit at Arizona Science Centre which is similar to the way Eye of Judgement works. It also uses cards with special barcodes, except instead of monsters the cards bring up video cubes which can be moved and rotated by doing each action with the actual card. The exhibit is by a company called Total Immersion.

From looking into what is already available on the market it can be seen that this is an interesting area for investigating. The software found uses augmented reality very well, but the use which this project will be using is slightly different in the fact that this project is trying to use augmented reality on a camera which is actually moving. This will be where it may differ from the games already available. This may also make the game harder to code for as objects are moving towards the camera unlike the games show above where the camera is at a se distance. The games above show that there is an interest in augmented reality as it is an upcoming technology.

## *2.7 Conclusions of Literature Review*

After finishing the literature review the project has a better understanding and grounding. From carrying out the literature review a few conclusions can be made. They will now be explained in this section.

Augmented reality has been fully understood and will be used in an appropriate way. Augmented reality is the use of computer generated images to overlay a picture of real life, and can be done in various ways. During this project it will be used in a real-time environment.

Augmented reality has been used in a wide variety of industries from operating rooms to jet fighters. Even though there are many different industries using the technology, it is used in a similar way, technologically. This shows that augmented reality can be used across industries, therefore it can then be used in the game program this project is using.

Java is being used in this project so finding out how to use image processing works in java. This would be useful to create different effects on the screen, although the initial reason for researching this are was to find out how to add objects into the image. This was not quite found out but another door was opened to add effects onto the screen.

Looking at how to control robots gave a good insight into how they used to be programmed, with each robot having its own programming language. It also said about how nowadays C++ and java has been used, so means these can be used in the project. There were also some good tips for how to go about programming for robots.

Finding out what has been done previously with the SRV-1 provided some useful insight into how it can be used. It was also good to know that image recognition has been used with the robot. Another point of interest is that augmented reality has never been used with the SRV-1 so means it is an interesting topic to look into.

Finally with a look into augmented reality being used within computer games, it showed that there is a definite market for AR games. Games using augmented reality generally use some kind of card with special barcodes on them, this idea could be used within this project with the use of image recognition. Also the game in this project will be slightly different from previous games as the camera will be physically moving where other games use a stationary camera, so this may be slightly harder to program for as depth will have to be considered.

To conclude, the literature review has been a valuable piece of research and has

provided new ideas and reinforced previous ideas too.

# 3.0 Methods

This section of the report will explain what was carried out for the actual piece of primary research. This will incorporate all of the design decisions which were made during the course of the project. There will also be a discussion of why these decisions were taken. There will be a look at how the implementation was carried out and any problems that were encountered during this stage of the project.

## *3.1 Initial Designs*

During the creation of this project some designs for the implementation needed to be looked at, this was where the first look into what would actually be created in the game, as well as how it could be done.

### 3.1.2 How to Create the World

One of the main design decisions which needed to be taken before any coding could begin was; what technique is to be used for creating the actual game. From the literature review there was found to be many different techniques which have been used within other industrial settings. These were things such as using a method of image recognition to make 3D images pop up when different things are recognised by the software. Another technique found during the literature review stage was that of post production overlaying images. This was found within the movie industry.

The technique chosen for this project was; the creation of a 3D world which held all the position values for the 3D models. The 3D world would then have a textured background which would be the feed from the SRV-1 robot's camera. This was chosen as it would be the quickest and easiest way to create the world, as it would not need any image recognition software to be implemented. One of the biggest reasons for image recognition not being used was because from reading into how it would work. It was discovered, from looking into a piece of software for image recognition, that when this software was used it would mean that the feed would be running very slow, 2 or 3 frames per second (FPS) compared to the normal 30 FPS. As the program being created for this project is a game it is desired to have a quick running program so image recognition was therefore taken out of the initial designs.

### 3.1.2 3D Objects

When designing the game it was known that it was going to be incorporating 3D models over a real world backdrop but how these objects were to be created needed to be decided. To design these models it was important to know what models were needed. As the game would be featuring a wide variety of objects each one would need to be thought about. The game would have the following objects in it; pickups, targets, missiles and walls. As each object is very different they would need to be very different looking so they are not easily confused for one another by the player.

The initial design for these objects was, for them to be created within Softimage XSI, this was the program chosen to do this as it had been used before by the programmer. A

downside to using this program is that when exporting the models they sometimes do not get imported into the IDE the way they looked in XSI. This was discovered through other projects worked on. However the other projects were all using Microsoft Visual Studio, so it may be a problem with that rather than XSI exporting. This project will be using Netbeans with Java so may not be affected in the same way.

### 3.1.3 Envisaged Functionality

In order to know how to go about programming the game the functionality which is expected needs to be written down to make sure everything is taken into account, rather than finding something is missing after the implementation has taken place.

As mentioned before this project is about creating a game using augmented reality in a Mario Kart style. Therefore a look at what features Mario Kart has would be a good place to start. Mario Kart is an arcade racing game which features most Nintendo characters. Some of the best features of the game that could be used within this project are; weapons, racing and battle mode.

The weapons featured in Mario Kart are things such as turtle shells which can be shot forwards or backwards at other competitors within a race, other weapons include; banana skins which stop any kart for a short time if they drive over it. A lightning bolt which shrinks all other racers allowing the player to drive over them. The mushrooms is another weapon, it gives the player a short power boost for a few seconds. As this game is going to be a stripped down version of Mario Kart it would be good to have at least one of these type of weapon, most likely a rocket style weapon as it could be used to

shoot at anything.

With the main feature of Mario Kart being racing it would be a good feature to implement in this project. However this would involve a lot of AI programming which could be very complex. Also with the game using augmented reality it may be difficult to create a race in the real world, as other robots would need to be used.

Instead of making the game into a race it could be made into a pickup and shoot other objects in the 3D world. This would be easier to program and would also be better for the real world as only one robot would be needed as all the objects are in the virtual world. So the main functionality for this project is to have pickups in the virtual world and also have targets for the player to shoot at, when the player picks up a pick up, they will be given a rocket to shoot at the targets, each pickup will add a rocket to the player inventory. When a rocket is shot it will take one away from the player's inventory, meaning the player will need to find pick ups to shoot all the targets.

This functionality described above is similar to the battle mode of Mario Kart, however in Mario Kart the player is trying to shoot at other players or AI. The AI part of this would be good to have in this project but this project is more geared towards how the augmented reality system works so AI was not going to be implemented.

To summarise the envisaged functionality. This project is using augmented reality techniques to create a game which is like Mario Kart. The technique which will be used is somewhat like post processing technique like that used in the movie industry. The game will be created in Java 3D and will have a textured background which will be

created using the camera feed coming from the SRV-1 robot. Objects will then be put into the 3D world, therefore giving the impression of the 3D objects being overlaid onto the real world image from the camera. This mean the objects are being placed manually by the programmer and also image processing is not being done on the camera feed, the reason being the feed would be running very slowly if it was using image processing. The player will be able to move around the 3D world and in turn they will be moving the robot in the real world, so controls will need to be added for both the 3D world and the robot in the real world. The player will have the ability to pick up objects by moving towards them, this means there will need to be a collision detection set up on the pickup objects for when the camera is at them so there will need to be a comparison between the camera position and the 3D object's position. The player will also have the ability to shoot a rocket to kill the target objects so again there will need to be collision detection set up to check the position of the rocket and the target object. So the game should be somewhat like an augmented reality Mario Kart with some features taken out.

## *3.2 Implementation*

This section will give an insight into what was actually done during the implementation stage of this project. This section will be split into many sections detailing how each part of the program was constructed. There will also be details of any problems which were encountered during the implementation.

### 3.2.1 Setting Up the World

As Java 3D was a new language to the team, a few tutorials on how to use it were found and looked at to give the team a quick insight into how to start up a project in Java 3D. From looking on the internet many different techniques can be used to create a 3D world using Java 3D. In this project a method shown on the javaworld website which basically created a simple universe which could allow for a viewing platform to be created. A canvas was also created which is a window where the universe can be looked at through. This was fairly easy to set up as the website had a good tutorial walk through showing how to do it.

### 3.2.2 Adding Content to the World

The next step was to create content in the 3D world, this was done through using the box class within the geometry class. This was done relatively easily as the box was very easy to create through a few lines of code. The box appeared in front of the camera however the position of the box should not have been where it was due to when creating the box it was thought that the position values being passed to it, however when changing them it became clear that this was actually the size values as the cube would change size. After looking into the Java 3D API it looked as if there was no way of positioning the objects within the 3D world. This put the project into limbo for a while because without being able to move objects around the game would not function at all.

After looking into the API for moving objects around, it was also looked at to find out how to move the camera around, this also seemed that the camera could not be moved. So after finding this an internet search was carried out to find out how others were able to move objects and the camera around the world.

### 3.2.3 Adding Transform Groups

This internet search found that in order to move objects around in Java 3D they needed to be attached to a transform group. A transform group is a class that handles all the positioning of objects within the 3D world. With a transform group the position of an object can be set by using the setTransform() method. This has a transform3D passed to it, which is a matrix with position details held within it. The transform3D is what actually sets the new position of the object with the setTranslation() method which has a vector3f passed to it which has the x, y and z values of the new position.

From the search it was also found that the viewing platform can be changed to move the camera around the world however when trying to do this it was not moving the camera at all so the decision was taken to move the objects around the camera rather than moving the camera around.

### 3.2.4 Moving the Objects Around

As mentioned above trying to move the camera around was more hassle than it was worth, so to combat this, the objects themselves would be moved to simulate the camera

moving. This technique is much more complex than moving the camera as the camera is only one object compared to every other object in the world as there could be limitless other objects. Originally there was only one transform group, this handled the movement of the one object within the world. When a new object was added into the world it was in the same position as the first one. This was not desired as when the new object is supposed to be put in a different position giving the game a bit of variety.

It was later found that transform groups can have children added to them. This meant that if there was a main transform group then other transform groups could be added to them as children. Then when moving the main transform group it would move all other transform groups in the same way. This was desired as all the objects should be moving in the same way as the camera is what was being simulated in this game.

As there had been a few different problems with the implementation it was decided that the programming was the main priority so 3D modelling with SoftimageXSI was taken out of the development. This was because it could potentially take a large chunk of time to do and might not even work with Java 3D. If there was any time left at the end of the project then 3D models could be created to add polish to the game, but at this stage of the project functionality had to be the number one priority.

### 3.2.5 Orientation of Objects

The transform groups mentioned above were used when the movement was just moving on the x and z axis forward and back, meaning that moving just one transform group could work because it moves all of the child transform groups by the same amount.

However when the control style was changed to allow a simulation of panning of the camera, rotating left and right, this was giving unwanted results.

To achieve a look of the camera panning around some mathematics were needed. In order to find the new position of the objects being rotated around the camera, Pythagoras theory was used. This was used to find the distance of the object from the camera position for example to get the hypotenuse. The angle was then found from getting the inverse cosine of the z position of the object divided by the hypotenuse found above. Initially this was only done for one of the objects and it seemed to work so it was used on the main transform group. However after a while of using this it became clear that something was wrong. When moving over to the second object it was found that this object was moving way too much for how much rotation that should have been applied. It was later discovered that because the first object was the one being used to create the rotation that meant the rotation of the other object was as if it was in the same position as the first so the movement was not what was desired.

In order to cure this, transformations were taken away from the main transform group and added to the each child transform group. This involved a slight rework of the code but it made the objects move in the desired way. In order to make sure each object moved in the desired way there were hypotenuses set up for each of the objects. This ensured that all objects were being held at the correct distance from the camera position. To begin with there was only one rotation variable set up, however this caused the objects to jump quite a distance when the rotation was called, even the original object was not moving correctly. This was due to the rotation value didn't change as the forward and back buttons were pressed. Initially there were no methods to check the

new angle using the hypotenuse so the angle used was just the old rotation of the object so it threw up some strange results, for example the position of the object would jump quite a lot after a translation either forward or back. The rotation did work as expected if there was no translation forward and back. This problem took a while to figure out and was quite tricky to implement the fix which was to create the methods to work out the new hypotenuse and angles involved. To find the hypotenuse Pythagoras theory was used, and in turn this hypotenuse was used find the new angle of the object compared to the camera position. This made the movement of the object to be as expected, therefore there was no jumping around when translation occurred. There was a minor problem with this however, when the object was in a negative x position when rotating the object would jump over to the positive position rather than stay negative. This was easily remedied by adding an if statement to check if the x position of the object was negative and if it was, it then made the hypotenuse of that object negative.

### 3.2.6 Controls

The controls were added into the game at an early stage, originally just to check the controls actually worked they were added just to change variables and print out the new values of each variable. All the methods for getting to key commands were added these were; keyTyped, keyPressed and keyReleased methods, all of these methods need to do something or else the program will get an exception. The keyPressed method was used for the changing variables, the keyTyped and keyReleased methods were not used but they needed something in them just to run the program without an exception so a print command was added just so it would do something.

When the controls were first implemented they didn't work they did not print anything out. To check whether it was a problem with the controls a print command was set up outside the control methods. The print command was set up after the variables were initialised. This worked so it proved that there was something wrong with the controls. It was discovered that in order for the program to know that input from the user will be taken a command listener needs to be set within the program, this was added to the canvas in this case. With the command listener added to the canvas, the controls worked, with the variables changing and being printed out.

With the controls working, the proper controls for the program were added rather than testing to make sure input was working. At the beginning of the project the controls were set up just to move the x and z positions of the objects. This helped with finding out to make sure everything is being moved around the screen like expected. This worked well for testing the movement of onscreen objects, at the beginning the objects were not moving around the screen even though the position variables were changing. This was where it was discovered that transform groups were needed in order to move the objects around on screen. So once the transform groups were added into the implementation the objects were moving around the screen.

With movement working it was then desired that when the player presses the button to move left and right they should rotate around the camera position rather than just moving the x position along by 0.1, as it had been doing. To do this methods called rotateL and rotateR were set up, these methods rotate the objects around by using the hypotenuse and angle found described earlier. The methods either add or subtract 1.0 from the angle variable this is then used in two equations which find the new x and z

position given that angle, it uses the sine of the angle multiplied by the hypotenuse to get the x position and the cosine of the angle multiplied by the hypotenuse to get the z position of the object. These two positions were then used in a method called setPosition which changes the position of the object.

The setPosition method is simply a method which places the object in the world it is passed four floats, x, y, z and rotation. These variables are then used to put the objects in their position. The x, y and z variables are self explanatory being the objects x, y and z position variables, however the rotation variable needs some explaining. Rotation is basically the used to rotate the object on its y axis. This is to make the object look more realistic when moving as it actually rotates on its own local coordinate system. The rotation is updated inside the rotateL and rotateR methods. So with every use of these methods the rotation is added to or subtracted from by 1.0. This means that every time an object is directly in front of the player the rotation will be the same. This may not be exactly true compared to how it works in real life, as it is quite difficult to work out due to the objects moving are not the camera like it is in real life.

Another very simple mistake made during the implementation of the controls was, when the controls were controlling the x and z positions of the objects they were moving the objects left, right, forward and back like they were supposed to but when realising the controls were supposed to be simulating the control of the camera this means that all the controls had to be reversed as, when the camera moves left the objects should be moving in the opposite direction. So to make the controls work as they were supposed to the commands that were put on the keys were simply reversed and this gave the desired result.

The final control added to the game was to enable the missile to be fired, this was placed on key close to the movement keys for ease of access. When the button was pressed it was supposed to allow the missile to move on its own. However it was found that the missile was only moving once, each time the button was pressed. This was not the desired effect as the missile should move in the way a real life missile moves. To make the missile move in this way there needed to be a loop set up so a for loop was set up within the button press, this made the missile move to the correct position but it did not show the path of the missile which is what needs to be seen. As this did not work a different method had to be tried. This involved creating a boolean called missileFired when the button was pressed it set this to true and within the main program there was a while loop set for while the program in running. A nested while loop was added to this loop to check whether missileFired is true. If it is true then the position of the missile is added to every loop which shows the path of the missile. Due to the use of a while loop this could be infinite so an if statement was used to stop this from happening. The if statement stopped the missile from going too far. It checked if the missile z position was less than -50, and if so it would set the missileFired Boolean to false and set the position of the missile back to the origin.

The final controls used in the game are:

W; move forward.

S; move backwards.

A; rotate left.

D; rotate right.

F; fire missile.

### 3.2.7 Collision Detection

Collision detection is a vital part to any game and this one is no different. As this game involves both picking up objects and shooting a missile at objects, there needs to be two different types of collision detection, one for pickup objects and one for missile hitting objects.

The first of the two collision detections set up was for pickups. This basically checks the position of the objects compared to the cameras position. To compare the two positions if statements were used which check each objects position against the origin. The if statements used check each of the variables of the object x, y and z. For each of these there are two checks happening, these are to make sure the position is between two points as the object is not just one point in the world. Making sure the object is between two points represents the width and depth of it.

When a collision between the camera and a pickup is detected the boxesCollected variable is added to, then the y position of the object collided with is set to 100.0 this is to give the effect of the object disappearing. The object being at 100.0 on the y axis also has another function which is to make sure it can never trigger a collision again, because the collisions between the objects and the camera only occurs if the position of the object on the y axis is less than 50.0, so at 100.0 this will never trigger the collision. This type of collision also gives the player another missile to use, as it is a pickup after all.

The game was always going to have a shooting element so collision detection was needed for checking targets against the position of the missile. This was done in a similar way as the collisions for the pickups but as the missile is actually moving it means that it cannot be just a check against the origin.

The collisions between the missile and target objects are found by checking the position of the missile compared to the position of the target. An offset is added to the collision to account for the size of both objects. This also uses the y position of the objects to ensure an object cannot be collided with more than once.

The collision response of the target objects are different to that mentioned above for the pickups. When the missile collides with one of the targets the boxes collected variable is added to, the y position of the target object is set to 100.0 like above. To make sure only one target can be taken out at a given time when the collision occurs the missileFired variable is set to false therefore the movement stops, the position of the missile is then set back to the origin. A missile is then taken off away from the amount of missiles the player has left.

```
if ((posValue.x>missilePosValue.x-1.0d) &&
        (posValue.x<missilePosValue.x+1.0d) &&
        (posValue.z>missilePosValue.z-1.0d) &&
        (posValue.z<missilePosValue.z+1.0d) && posValue.y <50.0d)
    {
      boxesCollected++;
```

```
            posValue.y = posValue.y+100.0d;
            setPosition(getPosition().x, getPosition().y, getPosition().z, getRot());
            System.out.print("boxes Collected = "+boxesCollected);
            missileFired = false;
            missilePosValue.z=0.0f;
            setPositionMissile(getPosMis().x, getPosMis().y, getPosMis().z, rot);
            missiles--;
        }
```

This is the collision detection for the missile as described above.

### 3.2.8 Texturing the Background

In the design stage of this project a robot was supposed to be being used, however the project was not running on time so as a result a decision to not use the robot was taken. Instead of using a real life robot it was decided to use a kind of simulation. The simulation took the form of a textured background. This was seen as a simulation of the real world robot because the robot would be showing a live camera feed to the real world, but this would basically just be a texture but a changing texture. In order to simulate this a background was created which used a texture of a 360 degree picture, hence with the picture being 360 degrees this meant that there is a view which can be viewed at any angle. The texture was created from the file of the picture using a texture loader, an appearance was then set up to hold the texture in. A set of texture attributes were also needed to set how the texture would be applied, in this case it would be

replace which overwrites any previous texture used. The material used for the appearance also needed to be set so the texture would look like it does in the file. The appearance was then used when creating the background, which was just a box. The box for the background was set to the size 180.0, 23.0, 0.5. This was to insure that the background would always be filling the screen.

When the texturing was first tried out it did not work. The appearance of the box was different but the texture didn't appear on it. After a look on the internet however it was found what was wrong. The box needed to be set up for texturing, the texture coordinates needed to be added to the box. To add texture coordinates to the box the normals and generation of texture coordinates needed to be passed to the box during creation and this was done with the use of a different box constructor. The background then had the texture on it and could be used in the game.

The background was set up to be 180.0 on the x axis because it would work out easier when movement was concerned. When the player is moving around the world the background will also be panning round to give the impression that the background is actually a feed from a robot in the real world. When the player moves forward and back in the game the background has also been set up to move back and forth, however when an object gets far enough away from the camera they disappear so to try to combat this the movement of the background has been set to be very little compared to the other objects, because the background is very close to the point of disappearing, at the start. The reason for this is because if the background is closer to the camera than other objects the background will be blocking the other objects from the players view which is not wanted as it would make the game very hard to play.

The game worked well with the background in it, a problem occurred though when the player turned 180 degrees the end of the texture was found at both sides. In order to make the background look like a continuous texture another box was set up with the same texture and placed at -360.0 on the x axis. Doing this made sure there was a continuous texture when moving left. Though when the player had done two full rotations the edge of the texture was still there. So the same problem was there, just taking longer to get to. The second attempt to cure this was to make the two backgrounds dynamic. This involved using if statements to set the position of the background depending on how far it is off the screen it is goes over 360 it gets moved -360 and if it goes under -360 it adds 360 to the x position, this is done to both the background boxes so it means there will always be a continuous background.

```
if (bc.getPosBackG().x >360)
        {
           bc.backGPosValue.x = bc.backGPosValue.x-720.0f;
           bc.setPositionBackG(bc.getPosBackG().x, bc.getPosBackG().y,
bc.getPosBackG().z, bc.getRot());
        }
        if (bc.getPosBackG().x <-360)
        {
           bc.backGPosValue.x = bc.backGPosValue.x+720.0f;
           bc.setPositionBackG(bc.getPosBackG().x, bc.getPosBackG().y,
bc.getPosBackG().z, bc.getRot());
        }
```

This is the piece of code to make the background position change making the background continuous.

### 3.2.9 Game Logic

The logic of the game is basically what makes the game a game. The logic is like a set of rules that has to be followed to allow the player to progress through the game. For this game the logic is relatively simple compared to big games. The logic of this game is somewhat based on Mario Kart, however as the game is less complex than Mario Kart only a little of the logic of it has been used.

The objective of this game is very simple. The player has to either collect or shoot a set of boxes which appear on screen. It is totally up the player to choose whether to shoot the boxes or collect them. Although to start with the player is limited to only three missiles. When the player shoots a missile no matter whether the missile collides with a box they loose a missile from their inventory. When the player collects a box they are rewarded with another missile.

At the beginning of the game there are five boxes set up but the player only has three missiles. So in order to complete the game they need to collect at least one box giving them another missile. In order to keep track of how many missiles the player has each time a missile is shot one is taken off the amount of missiles left. To make sure the player cannot shot a missile when they don't have any an if statement was added to the

key to control firing the missiles, this if statement checked to make sure the missiles variable was greater than zero and if so it would allow the missile to be fired.

If the missile does not collide with any of the boxes it could have possibly kept going forever. This would have meant that the player would not be able to use the missile again as the way the missile is set up is to only use one object for all missiles just being reset to zero when a collision occurs. However if no collisions occur the position is not set to the origin, so the loop would not be broken and the missile would continue forever. In order to stop this happening the missile has been set to only go to a certain distance before resetting back to zero. This was done to simulate a real missile running out of fuel. It also means the player cannot only turn and shoot the missile, because if the missile could be shot to a distance which was in range of all the boxes the player would not even need to move forward and back, just turn and shoot. This would make the game very easy and would not give the player the best experience. This was a reason for only giving the player three missiles at the start of the game too, meaning they would have to move if they wanted to complete the game.

To finish the game a check in the main program for how many boxes have been collected, takes place. As there are five boxes the check is for when boxes collected is greater than or equal to five. If this returns true the program then sets a boolean called running to false. This then trigger a text display on the screen to let the player know they have completed the game. This text display was done using the same method as the heads up display, which will be looked at in the next section.

### 3.2.10 Creating a Heads Up Display

A Heads Up Display (HUD) is basically an overlay on the screen which is used to give the player information which is useful to them. In other games HUDs are used to let the player know what their health status, ammo, map and many other helpful pieces of information.

In this game a HUD has been set up to let the player know how many boxes they have collected and to keep them informed about how many missiles they have left. To create an overlay with text on the screen a new class had to be used. This new class extended on the Canvas3D class, the new class created was called OverlayCanvas the main difference between the two classes was the postRender method. Within the new postRender method of the overlay class the 2D graphics are being changed. The 2D graphics are being set to display a string. The string was set and position data was also passed an x and y value, the origin of the 2D graphics is the top left corner of the screen so x values are positive to the right and y values are positive going down.

When the idea of a HUD was first implemented the standard canvas class was being used. The get2DGraphics method is available in the standard class so when the game starts this was used to display the amount of missiles the player had. However this did not work. It was later discovered that this was due to the postRender method mentioned above as in the canvas class it did not update the 2DGraphics.

After the new OverlayCanvas class was created with the new postRender method in it, the text was set within this class to test the displaying was working. With this working a

new method was written to set the variables for displaying and setting the position too. This new method passes a string and two integers which are then used as the text to be displayed and the x and y positions of the overlay. When using this the methods were set to change the display only when a missile has collided with a box. This seemed to work quite well however it took two lines per collision detection to add this into the game. As there was ten collision detection statements this would mean twenty lines were needed to be added just to change the small it of text.

It was decided to put this method call into the main game while loop as it would only need two lines of code as they would be updating every loop of the game this would ensure that no change to the game state would be missed. The text is displayed in the top left corner of the screen and consists of the amount of boxes collected and the amount of missiles left for the player to use

```
bc.getCanvas().setVariables("Boxes Collected: "+bc.getBoxesCollected()+" Missiles: "+bc.getMissiles(), 30, 30);
bc.getCanvas().postRender();
```

This is the creation of the HUD and is inside the main game loop.

The one downside to using this in the main game loop is that it causes the text to flash slightly, this is because it is updating the 2D graphics every loop.

## *3.3 Testing*

This section of the report will detail how the program was tested for its functionality. There will also be a quick description of any error and how they were fixed. The program was continuously tested during the development.

### 3.3.1 Test Cases

Before any development had taken place the functionality of the program was being designed and from these designs a thought about how the functions could be tested was needed. From knowing at the beginning of the design stage that the game would be user controlled. Therefore test cases needed to be set up to make sure the controls work the way they were supposed to.

Test cases were written up for each of the controls during the implementation. Each test case was written with the expected result of each button press and the actual result. As the controls were always being slightly changed the test cases had to be changed slightly with each iteration of the control method. There were very slight derivations with the expected results for the controls early on in the project, for example the objects were not moving in the correct direction, for trying to simulate the camera moving instead of the objects. This was easily fixed as it was just swapping around positive and negative signs.

Later in the development the controls were changed in order to allow rotation from the

cameras position. This was found to work as expected but there was a large jump in the position of the objects that should not have occurred. This was found no to be the fault of the control system but instead a slight fault of the way the new position was calculated. The position was calculated by using the rotation value, this value was set up in degrees but the math methods that were used take in radians so the values were way different than they should have been. This was easily changed due to the toRadians method in the math class which takes in the degrees value and converts it to radians. This took a time to figure out that this was the reason for the strange movements. Therefore some time was lost from trying to figure it out.

The actual controls were working in the desired way so in order to check methods for moving objects around the world were working correctly the controls were used to check these were working. To make sure the positions were being altered correctly a print command was set up to show the new positions of the objects. The set position methods worked as expected and as it was known to be working well it would mean that if another method was being used in conjunction with set position it would be the other method that would have a fault if an error were to occur.

The next method to be checked was the rotation method. When the method was working converting degrees to radians it was then checked that it was actually working by moving the object round by one degree at a time. When the rotation was being used changing between left and right it was rotating correctly however when going between forward and a rotation the object would jump slightly out of position. After a while of trying to fix this it was realised that when the movement forward was happening because the angle between the object and camera was changing. To fix this two new

methods were created to find the new hypotenuse of the triangle between the object and camera. This hypotenuse was then used in a method in order to find the angle between the object and camera. This worked to resolve this problem as when the object was moved forward or backwards these methods were called and updated the hypotenuse and angle values. However this threw up a new issue. When the x position of the object was positive the rotation worked fine but when the forward or backwards buttons were pressed the x position snapped back to the positive value. The reason for this was found to be the hypotenuse method as the x position was being squared it would always be positive so an if statement was set up to make the hypotenuse negative if the x position was negative. This cured the fault, stopping the jumping to positive.

Another set of methods that needed to be tested were the collision detection methods. As collision detection was one of the main parts of making the game work properly it needed to be checked thoroughly. To make sure the collision detection looked correct different variables were used for the checks. Initially the variables were too large and collisions were happening when the objects were nowhere near each other. To try to make the collisions look better the variables were reduced slightly. The size of both object were also taken into account and used as the offset. However this did not give the expected result, this may have been due to the scale value used. Many variables were tried after this, after some trial and error the best fitting collision was having the offset at +-0.67 for the missile collisions. This made the collisions only happen when the object were actually touching visibly.

The logic of the game also needed to be checked with some test cases. To make sure the correct things were happening when the player done certain things within the game. The

main logic is updated inside the collision detection methods, with different things happening with different types of collision.

To check whether the variables were actually being changed during the collisions a print command was used. These were found to be working as expected. The other piece of logic to be checked was the logic for ending the game. This was set up to display a message to the user to say well done for collecting all the boxes when boxes collected was equal to five. This was found to work in the way it was intended. This was very important to check because it lets the player know they have completed the game.

## *3.4 Actual implementation compared to design*

this section will be describing the differences of the final product compared to the initial design plans. The reasoning for these changes will also be given. An implementation design will be given for the dropped features to show there has been some thought put into them.

### 3.4.1 Planned Implementation

Before any implementation took place during this project the designs were created. The design was to create a game using augmented reality with the use of a robot with a webcam to use the feed from to overlay 3D images onto. In order to display the 3D images on the overlay image recognition was going to be used. The game would have taken place on the computer but also in real life with the robot moved around in the real

world.

Objects for the game would have been created in Softimage XSI these objects would be things such as pick ups, targets, checkpoints and missiles. The objects would be imported directly into the game. Image recognition would then place these objects on the screen when certain real life objects were recognised by the system.

The game was to be like an augmented reality version of Mario Kart so features taken from this were; using weapons and checkpoints. These were the two best features to use as the racing would not really work with the real world augmented reality being used. Artificial intelligence was also not needed as racing was not involved.

### 3.4.2 Actual Implementation

As described earlier the actual implementation was created in Java 3D. The game was created with a textured background which could move around giving the player the impression of moving around in the world. There were objects placed around the world which were created from the Java 3D geometry classes. These objects were placed in the word manually through code.

The game was set up so the player could move around the world using the controls. However the players view point was not actually changing position. The position of the objects within the world was what was actually changing.

The objective of the game is to collect the five boxes that are spread around the world.

This can be done by either moving towards the boxes and picking them up by moving directly into them or by using the missiles to shoot the boxes.

An overlay was created to keep the player informed about their progress in collecting the boxes. This was the player's heads up display. The HUD also displayed how much ammo the player had left. The player could also collect ammo by collecting the boxes and would loose ammo by shooting their missiles.

### 3.4.3 Differences between Design and Implementation and Possible Further Work

The main Difference between the designs and the actual implementation was the fact that there was no actual robot used in the game at all. The reason for this was due to time constraints. The project was underway and due to some features not running the way they were supposed to the project started to fall behind and as the robot had not been used yet it was decided to cut the use of it from the project. The project was not started quite as soon as it should have been so when other things started not go as quickly as expected the robot was replaced with a much quicker technique to program.

Instead of using a real life robot a textured background was used this was created from the use of a 360 degree photograph. This background was used to give the impression of the real world, to achieve this the background was set up to move around the 3D world when the player was moving around. This worked quite well as a replacement for the robot, as it was a full 360 degree picture, this meant that the background seemed to be like the real world.

Without the robot there was no real world element to this project. This is one of the biggest regrets because the game would not only have been controlling the 3D world which was created but would have been controlling a real world object. However this makes the project extendable in the future.

This project has shown what can be done with Java 3D and gives a look into how augmented reality could be used. A way which a robot could be used in an extension of this project is; as there is already a textured background in this game, rather than having the texture being a static image from a file the image could be replaced with the camera feed from the SRV-1 robot. The background would also not need to be movable as the images are coming straight from the robot. The image feed from the camera could be saved to a file and be overwritten as the new image is taken. This file could be used to create a texture to set onto the background. This however could create problems with updating the texture of the background as when the texture is applied to the background in the current implementation it is done at the beginning and only once. This could be done during the main game loop to update the texture updating every loop. The background in this implementation is created during the initialisation of the basicConstruct class so therefor is not a global variable, but in order to update it in the main game loop it needs to be a global variable because it will be inaccessible otherwise. This could be easily fixed though by creating a class variable for the box which can be set to public so it could be accessed throughout the entire program. Making the variable public is not desired though as it could be accessed by accident when it was not supposed to. To protect against this a method could be set up in the basicConstruct which updates the texture so this will allow the variable to be set to

private so the method could be called within the main loop and not allowing access to the variable to change anything else which is the best way to do this.

As the robot was not used the controls within the game were quite easily implemented as only controls for the 3D world were needed. It should be relatively easy to add controls for the robot to be manipulated. The robot could be controlled through its own class this would make it much easier to use as it would be just a case of calling methods from the controls method of the basicConstruct class.

A change would most likely need to be made to the current controls system too. This is due to making sure the real world movements are in synchronization with the virtual world movements. An example of this would be if the robot is rotating it may only rotate by 45 degrees in the real world but in the virtual world the objects may have rotated by 90 degrees. So the correct amount of rotation, in the virtual world, would need to be checked in order to make sure the objects are appearing in the correct position relative to the real world feed. This is also true of the forwards and backwards movement. The robot may not be as quick as the movement in the virtual world. To try to make sure the correct amount is being added to the virtual world positions it may be of benefit to use benchmark tests such as time tests to see how long it takes for the robot to reach a point 10 centimetres in front of it. This would then allow the programmer to work out how much the virtual world needs be moved each loop. This could also be done for rotation, a test could be done to check how long it takes to rotate by 90 degrees, again this can be used to work out how much rotation is needed each loop.

Another thing which did not make it from the design to the implementation was image

recognition. The reason for this was because it was not going to give the performance needed to run a game. The robot would also have needed to be used for image recognition. It was felt that there was not enough benefit to be had from using image recognition. From the literature review it was found that one program which could be used to carryout the image recognition would make the feed only run at about 3-5 frames per second. Considering that the feed would normally run at 30 frames per second this image recognition would make the feed run very slow this would in turn have a bad effect on the game making it run very slow too. Due to finding this out it was decided to create the game using a 3D world with all the objects being placed manually by the programmer, this allowed collision detection to be carried out easily too as the positions are know from the beginning so comparisons of positions could be carried out from the start of the game.

If image recognition was to be used with this implementation it could be done, but performance would be lost. In order to do image recognition a program for image processing would be needed. These programs look for certain things within a 2D image. For using image recognition for this project, it could possibly be used to check for a certain colour on the image. It could look for a colour such as red in the image and if it sees the this in the image a 3D model could be placed in the 3D world at a position which is worked out by checking how far up the image the red pixel is. For example if the red pixel is 300 pixels up the screen the position of a box could be set to -15 on the x axis this would be because the distance of the object in the real world is a reasonable distance away. In order to trigger the recognition objects would need to be put into the area where the robot is in the real world. The best type of object to put in the real world would be a small LED this would give a bright glow so it should be able to be

recognised easily. Different coloured LEDs could be used for setting different objects in the world for example if there were to be two different objects being set up, one for pickups and one for targets, a green LED could be used to represent the pickup and a red LED could be used to represent the target.

One difficulty that may occur when using image recognition could be collision detection. This is due to the real world objects when they are not on the screen the 3D model could disappear. To stop this from happening it could be set so that when the game begins the robot could do a 360 degree turn to set up all the 3D models. This could be an initialisation of the real world within the 3D world. When the robot does this turn it could store all the positions of the 3D images. The game could then use these position values when the game is actually played rather than performing image recognition throughout playing the game. This would allow the program to run more efficiently, although this initial set up of the objects may take quite long. This method would make collision detection better as the objects are being represented inside the virtual world with position data. So this would make the game run far quicker and the collision detection would work as it does in any game.

In the designs for the game the 3D objects were going to be created using Softimage XSI, however during the implementation objects were created within the program using the Java geometry class. The reasoning behind this was quite simple. When programming started objects were set up through the geometry class, and with problems encountered during the run of the project there was just not enough time to create the 3D models to replace the boxes used in the game already. As the 3D models were not of high priority it was decided to just keep box models created in the program and focus on

the programming as it was of highest priority, and models were only really to polish up the final look of the game. 3D modelling also takes quite a bit of time it was taken out as time could not be taken away from programming.

If 3D models were to be used in the future iterations of this project it should be able to be done relatively simply. It would involve creating the models in XSI and exporting them in a file format accepted by Java, .x file extension should work. Then in the actual program a 3D model should be able to be created from a file, so it should be easy to import the file. However if Java is like C++ then if the 3D models are textured in XSI they may be imported either without the texture or with the texture but with wrong texture coordinates. It may be fixable by setting the texture coordinates when importing the models into the program. There may not be any problems with importing objects into Java so it could work fine straight away.

The game was supposed to have checkpoints in it in the design however these were changed to object pickups and targets. This was decided as checkpoints are generally used for racing games, where the player has to race other opponents to each checkpoint. So as there are no opponents in this game the player doesn't have anyone to race against so checkpoints are not really needed. Therefore these objects were made into pickups and targets so the player has something to do in the game other than just going to checkpoints. The pickups are put into the game from the start and can be collected in any order. The pickups are also targets and can be shot at to collect although if they are picked up the player gets an extra missile with each one picked up.

If checkpoints were wanted in the game it would be relatively easy to create them.

Checkpoints need to be travelled to in a specific order, to allow this to happen an integer could be set up to record how many checkpoints have been collected. Then when the objects are being placed in the world a check can take place on the integer holding the checkpoint information. All the checkpoints can be placed in their position but at 100.0 on the y axis, then when the integer is at the correct number for each checkpoint the y axis can be set to 0.0 on the current checkpoint. This would ensure all the checkpoints were only visible when they are supposed to be. And when a collision is detected between the player and checkpoint the integer will be added to by one and the y position of the checkpoint touched will be set back to 100.0. This ensures only one checkpoint is visible at any one time, meaning that the checkpoints are travelled to in the correct order. Which checkpoints in the game this does not mean that pickups and targets need to be taken out as this will add to the gameplay of the game, making the game more appealing to play.

# 4.0 Evaluation

This section of the report will detail the findings of carrying out this project. Through evaluating the program the main research question will be answered and also try to prove the hypothesis given in the introduction section of this report

## *4.1 Augmented Reality for games*

The main purpose of this project was to find out whether or not augmented reality has a place within games. As augmented reality has not been used much in the games industry this project was used in order to find out if augmented reality methods used in other industries can be brought into games to make a game different to other games already using augmented reality.

With games at the moment using augmented reality they only really use it by having a static camera set up and moving something, usually cards with barcodes, under the camera and a 3D model appears on screen. In other industries such as surgery, mentioned in the literature review earlier, a camera is used and moved around and recognition software is used to overlay 3D images onto organs on a screen. This was the type of thing this project was trying to create.

This project used a technique like that used in the movie industry with the use of post production editing. This was done by having a 3D world already set up with the 3D models placed in the world. A background was also set up to act as the real world

image. So the 3D models were being placed manually on the screen overlaying onto the background.

The way this game has been set up it simulates augmented reality. This is due to the background being textured with a 360 degree picture, it is a real world image being overlaid with 3D models like that done in the medical profession.

In this project image recognition has not been used, this has not affected the results found as the techniques used in other industrial settings do not all use image recognition. With this project augmented reality has been used like a heads up display like that used in military fighter jets. This has showed that a game can be created using techniques found in other industries.

The testing of the program was basically done to make sure the functionality of the program was what was expected of it. The testing took place when new features were being implemented into the program. This was an iterative method of testing as many features were added into the game so testing was carried out many times during the creation of this game.

After the final testing stage the process of evaluation the game to make sure it was fit to answer the research question. When evaluating the game it was basically played many times to check it was meeting the specification of the designs made earlier in the project. This project has taken place to find out about augmented reality and how it can be used in games.

Augmented reality has been around in games for the last few years however this game is using a whole new technique, with the camera moving rather than just cards being moved in front of the camera. As the game created for this project has used augmented reality like that used in surgery with the camera moving around and overlays being created over the screen it seems to have worked well as the game seems to be very playable.

This suggests that industrial settings of augmented reality can be used in different ways including in a gaming setting. As this project has not used an actual robot it means that there were less overheads which needed managed, however if a robot was to be used it could be done in such a way that the robot would be used in an area that is defined by the user. Due to the use of augmented reality, if image recognition was used the player could technically set up their own level designs. This is because with image recognition the objects in the world are found by the robot and initialised into the 3D world allowing new layouts every play through of the game. This version of the game created does not have image recognition in it but this could be done in future.

One problem that could occur whilst using the robot could be real world space. If this was used by someone who does not have a large area to use it in this could cause the game to mess up slightly with the positioning of the robot compared to the positioning in the 3D world. However with this version not involving the use of the real robot this is not really a problem. Everything is in the virtual world so the only problems which can be found is that of the programming that went into it.

One of the main problems with this game however is the fact that the player can go so

far towards the background that they can actually go through it. This happens because every time the player moves forward the background is coming closer to the camera. So if the player just continues to move forward they will eventually go through the background. In order to try to stop this happening all the objects have been set at distances that allow the user to reach before the background gets too close.

In order to truly find out the main result of this project in terms of did the main research question get answered by the piece of software developed, a recap of what was involved with augmented reality in different settings was needed. To do this the literature review was looked into again. From looking into the literature review again it was found that the way in which augmented reality has been used in different settings is relatively similar, however there are some slight differences. These differences are mainly the way which the graphics being used look. The main functionality being used is very similar, but with different content. With all of the augmented reality systems, a overlay is being projected over a real world image. This has been done in various different ways but in essence it is the same thing.

## *4.2 Answering the Research Question*

After what has been said above, it becomes clear that the research question has been answered. With all the industries using the same kind of augmented reality just in different settings this project has show that it has used the same augmented reality techniques in a new setting, a game. Therefore the answer to the research question is; yes a game can be created from the augmented reality techniques used in different industries. This is because when a comparison is taken between how the technology has

been used in all the different industries already using it compared to how it has been use in the game created for this project. the game has used the technology in the same way. This is allowing for 3D images to be added on top of a real life image in real time.

One slight difference however in the game created is that the images were not in real time. However the only reason for this was because of time constraints. This could be done in future iterations of the game. The image used in this game was a static image but it was from a 360 degree camera so it could be manipulated to look like it was a real feed. So even though there was no actual camera feed linked directly to the game, a simulation of this was achieved therefore giving the game the perception of using a camera.

## *4.3 Steps Taken in order for Program to Function*

It was discovered that during the implementation that some functions would not work before another method was created. There was various occasion when this came into account. Some of these stages will now be explained in more detail in this section.

### 4.3.1 World Creation

During the implementation the very first thing which needed to be done was the creation of the world. This is a very obvious point due to the fact without the world there is nothing being output on the screen except the output in the output window of Netbeans. The window for the displaying the world also had to be created with the 3D

world being attached to the window as nothing would appear in the window if the world is not attached to it.

### 4.3.2 Controls not Responding

When the keyListener was used to allow for controls to be used, the methods for handling key events were set with controls to allow the player to move objects. When the controls were tried out at first nothing was happening. Test cases were set up to just display a string in the output window of netbeans to check the controls to see if they were working. It was then found out they didn't. It seemed to be a problem with the controls. This was finally discovered that a keyListener needed to be set on the canvas being used, this allowed key events to be logged from when occurring on the canvas. So it was found that before key events were able to be used a keyListener had to be set on the canvas being used.

### 4.3.3 Transform Groups

In order to allow an object to be moved it was discovered that they need to be added as a child to a transform group. However an object cannot just be added into a transform group and just be moved around. Before any movement can occur the capability of the transform group has to be set. This basically is for allowing the transform group to change as it sets the capability to allow transform read and allow transform write. These two capabilities allow a transform group to change position, so they are not just static.

In order to set the translations for each object a strict set of instructions need to be followed. This is to set the position variables then use a temporary transform group to use set the position of the object on that then set the actual transform group for the object to this temporary transform group. This set of steps has to be followed with all the positions for the objects. These steps were put into the update function so within the set position method a quick call to the update function and the position is updated.

### 4.3.4 Rotation

With rotation there are a few steps which have to be in the correct order. When the player presses the forward or backward keys a set of calculations are set to check the hypotenuse and angle of the objects in relation to the camera position. This has to be done in that order, finding the hypotenuse and then finding the angle. This is because the hypotenuse is being used within the angle calculations. So if it was done the other way round it would be using an out of date hypotenuse.

These calculations are worked out in the forward and backward key presses this is because this is the only place that the hypotenuse is changing. As when the rotation keys are pressed the only variable being changed is the angle as the hypotenuse is the distance from the camera position to the object and this is constant when the player is rotating. Therefore it does not need to be worked out when rotating, it only needs to be accessed.

Another method which needs to be used within the forward and backward movement control is the collision detection method for checking for pick up of the objects. This

collision detection method has to be used as the last method in the movement control. This is to insure that the new position is being tested against and it also means that the check is only happening when it needs to allowing for the best performance of the program.

## *4.4 Summary of Evaluation*

The main and most important finding of this project was that the fact that the research question has been answered. With the use of a textured a background and 3D models the game has given a simulation of augmented reality with the background moving to simulate a camera feed. This therefore suggests the hypothesis is true; It will be possible to create a game using augmented reality methods used in industry. This is proven because the game has been created with the use of the same kind of techniques as found in different industrial settings as found in the literature review.

As mentioned before the game has not used a real life image to overlay onto, however a background has been used which simulates the real world. This set up has allowed the testing of an augmented reality system. The game may not use an actual feed from a camera but the way in which the background has been set up it makes the game seem like it has a camera feed attached to it, due to the background being a 360 degree texture and movable.

This game also shows that augmented reality has a place in the games industry. Some games have tried to use augmented reality in the past, but they were only really using recognition of a barcode to overlay models on top of a camera feed with the camera

being static. This game has added slightly more to this, it has used a moving camera to overlay images into a 3D world. As the image is not static it needs to update the images into a 3D world updating there position too. This is relatively easily done but positions needed to be worked out comparing the real world position of the camera, in order to make sure the objects are moving at the correct speed.

As this game did not use a real camera feed it could be argued that the game does not use augmented reality. This is not seen to be a valid argument however as the game is using a simulation of the real world meaning that, if someone was using the game they could be told that the game is using a robot but the robot is not in the same area as they were. The game has all the functionality within the 3D world so all that would need to be done is to change the background to a feed from a camera and the real world would be incorporated into the game.

From during the implementation it was found that some steps needed to be taken in a certain order for the game to work properly these included the key listener to be set up before the controls would work. Also transform groups needed to be set up before an object could be placed and in turn manipulated. Another set of methods that needed to be in certain order was the working out of hypotenuse which had to be done before finding the angles, as this would lead to the wrong hypotenuse being used.

# 5.0 Conclusions

This section will give a brief overview of what was involved in the project then will go on to look over what was achieved during the run of this project. It was also give a discussion of what can be taken from this project, giving a quick look at how the project could be enhanced in the future.

## *5.1 Overview of the Project*

### 5.1.1 Literature Review

This project was created to investigate the use of augmented reality. A literature review took place in order to find out various things, including; what augmented reality is, how it has been used in industrial settings and where it has been used. Augmented reality was one of the key focuses of the literature review, however it was not the only area being investigated. This project was being used to evaluate how augmented reality could be used in a gaming environment. Due to this, within the literature review it was decided to find out how augmented reality has been used in games in the past. This project was trying to create a game in a kind of Mario Kart style using augmented reality, so this meant the game would be involving the use of robots. Because robots were going to be used the literature review also looked into the use of robots in general, it also looked into the control mechanisms being used for robots.

### 5.1.2 Design Stage

After the literature review had been completed the designs for the actual primary research method were drawn up. The primary research method for this project was to create a game like Mario Kart using augmented reality. The designs were to create the game in Java3D using netbeans IDE the game would involve the use of a robot called the Surveyor SRV-1, this has a camera and is controllable through a Java console program. The design for the game included creating a 3D world in which would use the camera feed from the SRV-1 to set as a background so the player sees the real world in the game. Objects would then be placed in the 3D world making it seem like the objects are being overlaid onto the real world feed.

The initial designs for the game had functionality of; the player could move around the world and go to pick ups and checkpoints, they could also shoot at targets which would be dotted around the world. The player would have access to a weapon so they could shoot at the targets mentioned.

The controls to be used in the game would not only need to be attached to the virtual world, but also need to control the robot in the real world. This could be done if there is a class system set up so methods from each class could be called.

The next step in the design stage was how to create the 3D models for overlaying. It was decided to use Softimage XSI as it had been used before. In the design, the game was set up to have many different objects so each object would be set up to look different and therefore would be used for different purposes.

### 5.1.3 Implementation Stage

The implementation stage was the longest stage of this project as it is where all the project work has been carried out. The implementation stage took place soon after the design stage was over.

The very first thing to be done during this implementation was to create a world which would be used as a base to the whole game. This was done by looking into online tutorials to find how to create a basic world. Once the world had been created the content of the game needed to be put into it.

The first piece of content to be added into the game was a box this was used just to check that content could be put into the game. It was a box from the geometry class of Java. This box was used for checking movement in the world. It was discovered that the position variables of the box were non-existent. It was then found that in order to set positions of objects in the world, transform groups were needed.

Transform groups were set up to position the objects in the world. The transform groups needed to be set up correctly as they needed to be enabled to allow them to be changed. A transform group needed a matrix passed to in order to set the position of each object. This matrix was a transform3D.

Actually moving around the world was looked into. It was found the camera position was difficult to move, so in order to get around this the objects were moved around

instead. As there could be many objects in the world each object will need to be set for movement. This is one of the downsides to this method, as every new object being created has to be translated in the world when the controls are used. Therefore many set position methods had to be used to translate each object around the world.

The next step taken during implementation was to create methods which would be used to rotate the objects around the camera position. This allowed it to seem like the camera was moving around rather than the objects.

After the movement methods were set up properly the controls could be set to the proper methods. Controls had been set up earlier in the project but were only being used to set the positions by changing the positions on the x and z axis. So when the rotation methods were written the controls were switched to use these methods.

There were some problems with the methods used for rotating but when these were fixed the collision detection methods were set up. With there being two different types of collision needed two methods were set up to test collisions. Collisions for checking whether the object had been collected were set up, and another collision method was set up to detect collisions between the objects and the missile object.

With the collision detection set up the final piece of content to be added into the game was the background, this was used instead of a robot, this was because the project took longer than expected and there was no time to use the robot. The background was textured with an image which had been taken with a 360 degree camera so it gives the impression of using a real life camera on a robot. This was due to the background being

set up to move around the world. This background used gave the game a simulation of a real world robot without the need to have an area set up, in the real world where the robot would be moving around, therefore reducing overheads of the project.

When all the content was included in the game the actual game logic was set. This was things such as responses to the different collisions, keeping track of variables and setting the players ammo. The logic was a very important part of this game this was because any game needs rules to be set, otherwise it isn't really a game.

With the game now including logic the rules were set, however the player had no indication of the progress of the game. In order to allow the player to know how the game is going a heads up display needed to be set up. The HUD gave the player two details which were important to the completion of the game. These two details were the amount of boxes collected and the amount of missile the player had left to use. The HUD was used to keep the player informed about their progress at all times.

### 5.1.4 Testing Phase

Testing was done throughout the implementation stage of the project. All of the main features needed to be tested. During the run of the program, it was run at all times to check any new code was working. This meant that the project was becoming an iterative process. As each new feature was coded, it was then tested. If something was found to not work as expected the code would be looked at again and tried to find a new method was tried, then tested again.

After the testing was finished and the program was functioning as desired, a look at how the actual implementation differed from the envisaged functionality which was designed in the design stage. The main difference found was that the game did not use the robot. This was due to the lack of time available. As the game didn't use the robot, it used a simulation of a robot instead. The simulation was basically a textured background which was movable so seemed like a good replacement for the robot.

## *5.2 Conclusions Found*

This section will contain the sub conclusions found in each of the other sections including the literature review, primary research method and the evaluation section. These conclusions will be used to come to a final conclusion.

### 5.2.1 Literature Review

The literature review it was found that augmented reality had been found in many different industrial areas, this was useful as the augmented reality was being used in very different types of industries but the actual technology was very similar. This meant that from the different areas using augmented reality with the same technology it means the technology is very adaptable. This showed that the technology could be used for various different things, this was one of the reasons it was tried for this project, as it had not really been used in the way different industries used it, in a game.

Within games it had been found that the use of augmented reality was mostly just using

cards and specialist barcodes that allowed 3D graphics to be placed on a screen from a camera feed. In these type of game the camera was static therefore the only area being used in game was the area directly under the camera lens and this never changed, except the cards themselves. So this game would be using techniques from other industries as the camera was being set up in a dynamic fashion as it was attached to a robot.

A look into how image recognition can be used in Java. This was found to be able to use different image recognition techniques. The techniques found were to use things such as making the image negative, edge detection and altering the colour of the image. This is not very relative to this project however as the image recognition that was wanting to be used in this project was wanted to find different colours in the image. This would be used in the program to find LEDs in order to add 3D models into the game. Another area looked into when finding out about image recognition was a program found which carries out image recognition this program detailed the specification of how the image recognition was done. This program also detailed running specification of the processing when carried out on a feed. The program stated that the use of image processing would slow the feed down to about two to three frames per second compared to the 30 frames per second that it should be. As this recognition program would slow down the feed so much it was decided against using image recognition. This was because the game would need to be fast flowing so if the feed from the camera was going to be as slow as the 2-3 FPS then the game would not run fast at all.

The literature review looked into how robots have been controlled in the past. It was

found that robots in the past were very limited in terms of how they could be programmed. With each robot having its own programming language. It was discovered that now robots can be controlled through C++ and Java. This was useful to know as this project was going to be using a robot.

In the literature review a look at how the robot which is going to be used in this project, the Surveyor SRV-1 has been used in the past. It was found that the robot has had quite a few different uses in the past quite a lot of people have used it to recognise areas in the real world and used AI to find its way around. This was not quite as useful as first thought as there was not any code which could be looked at for the use this project was trying to use the robot for. With the robot not being used in the final implementation this look into the use of the robot was a bit unnecessary, however it was needed because the envisaged functionality was using the robot.

The final conclusion of the literature review was a look into how augmented reality has been used in games. This was quite a valuable investigation because it showed that augmented reality has never been used in the way it is going to be in this project, in a game before. Augmented reality has been used in games previously, namely in a game called Eye of Judgement on the Playstation 3. In this game there is a camera set up pointing down at a game board and uses card with special barcodes on them. This game recognises the barcodes and displays monsters on the board which fight against each other. This use of augmented reality is static in terms of the movement of the camera, where this project will be using a camera which moves around in the real world.

### 5.2.2 Implementation

During the implementation it was found that Java is a good development language to use as it is relatively easy to get to grips with. One down side to Java 3D is that objects cannot be placed without the use of transform groups. This was slightly difficult to understand at first because there was no direct manipulation of the position when the objects were created. When the transform groups were used for a while they were quite easy to work with.

As this game has not actually used a robot it seemed like it would not actually answer the research question posed in the introduction. With the use of a textured background however this allowed the game to simulate the use of  a camera. As the background was set to move and as it was textured with a picture of a 360 degree camera. The game has allowed a type of augmented reality with the use of this background being overlaid with some 3D objects. The game seemed to work in well with this background being used, it made the game seem like it was using a real world feed as it was moving around as it would be expected to if it was the real world.

### 5.2.3 Evaluation

Even though the game works well with the textured background it would have been better to use a real robot. As the true test of augmented reality would have taken place, due to it being actual reality that the objects would have been overlaying onto. However

even though the game is not using the robot it has answered the research question and proven the hypothesis proven this is due to the simulation of the real world that the background gives. It makes the game feel like it is using a real life robot as the background is continuous.

As the game has used augmented reality in a new way, to games, this project shows that there is a place for augmented reality within the games industry. Although the game is not using the robot it could be implemented into the game relatively easily with the feed from the camera being set as the background texture instead of the one actually used. The controls would only slightly need to be changed in order for the robot to be added to the game. With the movement for the robot being added to the controls for the game.

## *5.3 Final Conclusions*

This project has been through many stages during the development and this development has shown that augmented reality is a good way to add something different into a game. With very few games using augmented reality, this means that not many companies have used augmented reality in their games. This project could be used to show that augmented reality can be a very useful tool in adding something new into a game.

The techniques used in the different industries explained in the literature review have been put to good use in this game. Therefore showing that these techniques can transfer well to totally different types of industry. As these techniques are quite general they do fit well into all the industries looked into. This includes the games industry as the game

created has worked well with the use of augmented reality.

So the final conclusion to this project is that augmented reality works well with games. This game was in the style of Mario Kart but as the technology is quite open it could be used for lots of different genres of game. It could be used for racing games, shooting games, strategy games, role playing games, the list could continue forever. This project has given augmented reality a place in games offering a new way to implement it compared to other games already using it.

One downside however if the game uses a real robot is; when the robot is being used, there will be a constraint on it, with the actual physical space people would have where they would be using it. As a lot of people may play computer games in their bedroom, so most people don't have large enough bedrooms to be using robots in games. This may be one downside to using robots in games but as the robot is on a wireless network the game could be used practically anywhere, even outside.

## *5.4 Future Work*

This section the report will try to give an insight into how this project could be expanded in a future development. It will aim to give an explanation of why the project could be developed in this way.

### 5.4.1 Adding Robot Functionality

As this project has not actually used the robot it was supposed to be using, this could be done in a future project. Adding a robot could add more functionality to this project. This would allow the project to actually use augmented reality in the way it is supposed to be rather than just simulating it.

With the use of a robot in the game it can test out how well the game works in terms of how it is used in the real world. It can check to see how much room in the real world the game needs. If the game were to use too much space the game could be downscaled slightly in order to fit into real world confines that are involved.

### 5.4.2 Changing Game Genre

In order to test out how well augmented reality fits into the games industry a look into whether it can be used in different genres could be useful. The game created in this project could be used as a base in order to create a new game of a different genre. A

game could be created in a totally different genre, as this would be the best test of whether or not augmented reality can work with different games.

A possible genre that could be used is a role playing game (RPG). This would be desirable to look into as this type of game is very large in scale usually. So this would be a good test of whether augmented reality could work on such a large scale. If it was possible to have a successful RPG game using augmented reality this would mean that practically all games would be suited to this technology, as this is a very demanding genre of game in terms of size.

### 5.4.3 Adding Multiplayer

Mario Kart is a game which can be player by anyone really. There is also a very good multiplayer aspect to the game. This could be incorporated into the game created in this project. This could make the game very addictive and bring people together.

This could be a good way to find out if augmented reality can work using multiple robots all trying to work with each other in a network. The game would need to be set up to allow users to be playing at the same time, interacting with each other. The game would need to be used over a network with one player per machine, using a single robot per machine.

So a multiplayer game could be set up in the same way as Mario Kart works. One major thing which would need to be worked out is collisions between the robots. As the current game has the position of the player at the origin, this would mean the game

would probably have to be re worked to change the positions of the players so they could be checked for collisions between other players in the game.

# References


Anastassova, M. & Burkhardt, J., 2007. Automotive technicians' training as a community-of-practice: Implications for the design of an augmented reality teaching aid. *Applied Ergonomics*, In Press, Corrected Proof .

Andrews, S., 2005. Eyetoy: Play 3. *Sunday Times.* Nov 6, p. 32.

Anon. 2008a. *Eye of Judgment™ on PLAYSTATION®3: The Official Site.* [online]. Available from: http://www.eyeofjudgment.com/ [Accessed 13/11/2008].

Anon, 2008b. Total Immersion; Total Immersion Brings Fireworks -- Augmented Reality-Style - to Arizona Science Center's New Exhibit, 'My Digital World'. *Marketing Business Weekly*, , pp. 371.

Anon, 2004. Heads up displays. *III-Vs Review*, 17 (4), pp. 22-22.

Anon, 1994. Controlling robots. *The Industrial Robot*, 21 (6), pp. 40.

Anon. a. *Surveyor Robotics Journal*. [online]. Available from: http://www.surveyor.com/cgi-bin/robot_journal.cgi [Accessed 05/01/2009].

Anon. b. *Surveyor SRV-1 Open Source Mobile Robot.* [online]. Available from: http://www.surveyor.com/SRV_info.html [Accessed 05/01/2009].

Ashley, S., 2008. Annotating the Real World. *Scientific American*, 299 (4), pp. 27.

Day, B. & Knudsen, J. *Image processing with Java 2D - JavaWorld.* [online]. Available from: http://www.javaworld.com/javaworld/jw-09-1998/jw-09-media.html [Accessed 03/02/2009].

Haugh, R., 2005. High-tech tools transform THE OPERATING ROOM. *Hospitals & Health Networks*, 79 (1), pp. 52.

Moallem, M. & Khoshbin, R., 2006. An environment for programming and control of multi-robot manipulators. *The Industrial Robot*, 33 (4), pp. 254.

Pretlove, J., 1998. Augmenting reality for telerobotics: unifying real and virtual worlds. *The Industrial Robot*, 25 (6), pp. 401.

Tlale, N.S., 2006. On distributed mechatronics controller for omni-directional autonomous guided vehicles. *The Industrial Robot*, 33 (4), pp. 278.

# Bibliography


Anastassova, M. & Burkhardt, J., 2007. Automotive technicians' training as a community-of-practice: Implications for the design of an augmented reality teaching aid. *Applied Ergonomics,* In Press, Corrected Proof .

Andrews, S., 2005. Eyetoy: Play 3. *Sunday Times.* Nov 6, p. 32.

Anon. 2008. *Eye of Judgment™ on PLAYSTATION®3: The Official Site.* [online]. Available from: http://www.eyeofjudgment.com/ [Accessed 13/11/2008].

Anon, 2008. Total Immersion; Total Immersion Brings Fireworks -- Augmented Reality-Style - to Arizona Science Center's New Exhibit, 'My Digital World'. *Marketing Business Weekly,* , pp. 371.

Anon, 2004. Heads up displays. *III-Vs Review,* 17 (4), pp. 22-22.

Anon, 1994. Controlling robots. *The Industrial Robot,* 21 (6), pp. 40.

Anon. *Surveyor Corporation - Open Source Robotics.* [online]. Available from: http://www.surveyor.com/ [Accessed 05/01/2009].

Anon. *Surveyor Robotics Journal.* [online]. Available from: http://www.surveyor.com/cgi-bin/robot_journal.cgi [Accessed 05/01/2009].

Anon. *Surveyor SRV-1.* [online]. Available from: http://www.roborealm.com/help/Surveyor_SRV1.php [Accessed 05/01/2009].

Anon. *Surveyor SRV-1 Open Source Mobile Robot.* [online]. Available from: http://www.surveyor.com/SRV_info.html [Accessed 05/01/2009].

Ashley, S., 2008. Annotating the Real World. *Scientific American,* 299 (4), pp. 27.

Collins, J. 1997. *Gamasutra - Conducting In-house Play Testing.* [online]. Available from: http://www.gamasutra.com/features/production/070797/playtest.htm [Accessed 12/11/2008].

Day, B. & Knudsen, J. *Image processing with Java 2D - JavaWorld.* [online]. Available from: http://www.javaworld.com/javaworld/jw-09-1998/jw-09-media.html [Accessed 03/02/2009].

Evans, R. & Shoureshi, R., 1989. Gaining Control Over a Manipulator. *Mechanical Engineering,* 111 (6), pp. 76.

Geiger, C., Stoecklein, J., Klompmaker, F. & Fritze, R. 2007, "Development of an augmented reality game by extendinga 3D authoring system", *ACM International Conference Proceeding Series; Vol. 203: Proceedings of the international conference on Advances in computer entertainment technology; 13-15 June 2007. 2007; ACM International Conference Proceeding Series*Association for Computing Machinery, Inc ,

One Astor Plaza, 1515 Broadway, New York, NY, 10036-5701, USA, [mailto:SIGS@acm.org], [URL:http://www.acm.org/].

Haugh, R., 2005. High-tech tools transform THE OPERATING ROOM. *Hospitals & Health Networks*, 79 (1), pp. 52.

Jampolsky, A., Brabyn, J., Haegerstrom-Portnoy, G. & Schneck, M., 1994. Evaluation of heads-up displays for low vision applications. *Journal of Rehabilitation Research and Development*, 30-31 , pp. 318.

Kuikkaniemi, K., Turpeinen, M., Salovaara, A., Saari, T. & Vuorenmaa, J. 2006, "Toolkit for user-created augmented reality games", *ACM International Conference Proceeding Series; Vol. 193: Proceedings of the 5th international conference on Mobile and ubiquitous multimedia; 04-06 Dec. 2006; ACM International Conference Proceeding Series*Association for Computing Machinery, Inc , One Astor Plaza, 1515 Broadway, New York, NY, 10036-5701, USA, [mailto:SIGS@acm.org], [URL:http://www.acm.org/].

Moallem, M. & Khoshbin, R., 2006. An environment for programming and control of multi-robot manipulators. *The Industrial Robot*, 33 (4), pp. 254.

Myers, D.R. & Brown, M. 1998, "Interactive User Interface for Programming Robots", *Proceedings of the Third ASCE Specialty Conference on Robotics for Challenging Environments; Albuquerque, New Mexico; USA; 26-30 Apr. 1998; Proceedings of the Third ASCE Specialty Conference on Robotics for Challenging Environments*American Society of Civil Engineers, 1801 Alexander Bell Drive, Reston, VA, 20191-4400, USA, [mailto:journal-services@asce.org], [URL:http://www.asce.org], pp. 272.

Ong, S.K., Yuan, M.L. & Nee, A.Y.C., 2008. Augmented reality applications in manufacturing: a survey. *International Journal of Production Research*, 46 (10), pp. 2707.

Phillips, K. & Piekarski, W. 2005, "Possession techniques for interaction in real-time strategy augmented reality games", *ACM International Conference Proceeding Series; Vol. 265: Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology; 15-17 June 2005; ACM International Conference Proceeding Series*Association for Computing Machinery, Inc , One Astor Plaza, 1515 Broadway, New York, NY, 10036-5701, USA, [mailto:SIGS@acm.org], [URL:http://www.acm.org/].

Pretlove, J., 1998. Augmenting reality for telerobotics: unifying real and virtual worlds. *The Industrial Robot*, 25 (6), pp. 401.

Sasaki, T. & Kawashima, K., 2008. Remote control of backhoe at construction site with a pneumatic robot system. *Automation in Construction*, 17 (8), pp. 907-914.

Shin, D.H. & Dunston, P.S., 2008. Identification of application areas for Augmented Reality in industrial construction based on technology suitability. *Automation in Construction*, 17 (7), pp. 882-894.

Starner, T., Leibe, B., Singletary, B. & Pair, J. 2000, "MIND-WARPING:towards creating a compelling collaborative augmented reality game", *International Conference on Intelligent User Interfaces: Proceedings of the 5th international conference on Intelligent user interfaces; 09-12 Jan. 2000; International Conference on Intelligent User Interfaces: Proceedings of the 5th international conference on Intelligent user interfaces; 09-12 Jan. 2000*Association for Computing Machinery, Inc , One Astor Plaza, 1515 Broadway, New York, NY, 10036-5701, USA, [mailto:SIGS@acm.org], [URL:http://www.acm.org/], pp. 256.

Tlale, N.S., 2006. On distributed mechatronics controller for omni-directional autonomous guided vehicles. *The Industrial Robot*, 33 (4), pp. 278.

## Appendix A marioKartGame.java

```
package mariokartgame;


import javax.swing.JFrame;
import javax.vecmath.Color3f;
import javax.vecmath.Vector3f;

public class marioKartGame extends JFrame
{

   //this camera isn't really used
   public Camera cam;

   public static void  main (String[] argv)
   {
      //this is used in the while loop so it can keep looping
      boolean running =true;

      //this is the basic construct being set up
      BasicConstruct bc = new BasicConstruct("Mario Kart Game");
      //camera not used
      Camera cam= new Camera(0.0f,0.0f,0.0f);

      // this was used to check the camera was working but didnt
      if(cam.getCameraPos().x <= -1.0f)
         System.out.print("camera positioning working");

      //this adds all the content to the world
      bc.addBox(0.5f, 0.4f, 0.5f, new Color3f(0.5f, 0.0f, 0.0f), new Color3f(1.0f, 0.0f,
0.0f));


      //bc.setPosition(5.0f, bc.getPosition().y, bc.getPosition().z);
      //bc.addBox(0.5f,0.4f,0.3f,          new          Color3f(0.8f,0.0f,0.0f),          new
Color3f(1.0f,0.0f,0.0f));

      //adding 2 lights to allow the models to be seen
      bc.addDirectionalLight(new                Vector3f(-1.0f,-1.0f,-1.0f),                new
Color3f(1.0f,1.0f,0.0f));
      bc.addDirectionalLight(new Vector3f(1.0f,1.0f,1.0f), new Color3f(1.0f,1.0f,0.0f));
      bc.finalise();

      //setting the size of the canvas on screen
      bc.setSize(500, 500);
      //allowing the canvas to be seen
      bc.show();
```

```
        while (running == true)
        {
            /*if (bc.getPosBackG().x <-180.0f)
                bc.setPositionBackG(bc.getPosBackG().x+360.0f,          bc.getPosBackG().y,
bc.getPosBackG().z, bc.getRot());//backGPosValue.x = bc.backGPosValue.x+180.0f;
            else if (bc.getPosBackG().x >180.0f)
                bc.setPositionBackG(bc.getPosBackG().x-360.0f,          bc.getPosBackG().y,
bc.getPosBackG().z, bc.getRot());//backGPosValue.x = bc.backGPosValue.x+180.0f;
             */

            /*these if statements allow the background to stay continuous
            setting the position of the two bacground objects to make sure they
            are ovetlapping so no black screen could be seen*/
            if (bc.getPosBackG().x >360)
            {
                bc.backGPosValue.x = bc.backGPosValue.x-720.0f;
                bc.setPositionBackG(bc.getPosBackG().x,                 bc.getPosBackG().y,
bc.getPosBackG().z, bc.getRot());
            }
            if (bc.getPosBackG().x <-360)
            {
                bc.backGPosValue.x = bc.backGPosValue.x+720.0f;
                bc.setPositionBackG(bc.getPosBackG().x,                 bc.getPosBackG().y,
bc.getPosBackG().z, bc.getRot());
            }

            if (bc.getPosBackG2().x >360)
            {
                bc.backGPosValue2.x = bc.backGPosValue2.x-720.0f;
                bc.setPositionBackG2(bc.getPosBackG2().x,               bc.getPosBackG2().y,
bc.getPosBackG2().z, bc.getRot());
            }
            if (bc.getPosBackG2().x <-360)
            {
                bc.backGPosValue2.x = bc.backGPosValue2.x+720.0f;
                bc.setPositionBackG2(bc.getPosBackG2().x,               bc.getPosBackG2().y,
bc.getPosBackG2().z, bc.getRot());
            }

            //this is setting the variables used in the HUD and displaying it
            bc.getCanvas().setVariables("Boxes    Collected:    "+bc.getBoxesCollected()+"
Missiles: "+bc.getMissiles(), 30, 30);
            bc.getCanvas().postRender();

            //checking for when all boxes are collected
            if (bc.getBoxesCollected() >= 5)
            {
                //giving a game over message if they've all been collected
                bc.getCanvas().setVariables("You have collected all 5 boxes well done!!",
```

```
125, 250);
            bc.getCanvas().postRender();
            //setting running to false as game is over
            running = false;
        }
        //bc.canvas3D.postRender(/*"string",10,10*/);

        //checking for if the missile has been fired
        while (bc.missileFired == true)
        {
            //firing missile each loop
            bc.fireMissile();
            if (bc.getPosMis().z <= -50.0d)
            {
                /*if the missile gets far enough away the missile is put
                back to the origin, missile fired is set to false and
                one missile gets taken away from the ammo*/
                bc.setPositionMissile(0.0d, 0.0d, 0.0d, bc.getRot());
                bc.missileFired = false;
                bc.missilesMinus();
            }
        }
    }
    return;
  }


}
```

## Appendix B BasicConstruct.java

```
package mariokartgame;
import java.awt.event.KeyEvent;
import javax.swing.*;
import java.awt.*;
import com.sun.j3d.utils.universe.SimpleUniverse;
import javax.media.j3d.*;
import com.sun.j3d.utils.geometry.Box;
import javax.vecmath.*;
import javax.media.j3d.Appearance;
import javax.media.j3d.Material;
//import com.sun.j3d.utils.behaviors.mouse.*;
import com.sun.j3d.utils.geometry.Primitive;
import com.sun.j3d.utils.image.TextureLoader;
import java.awt.event.KeyListener;
//import com.sun.j3d.utils.behaviors.keyboard.*;

//import javax.media.j3d.BranchGroup;


/**
 *
 * @author Yosemite Sam
 */

//this is the class created to implement all the game content
public class BasicConstruct extends JFrame implements KeyListener {
    /**
     * Constructor that consturcts the window with the given
     * name.
     *
     * @param name The name of the window, in String format
     */

     /**
     * The SimpleUniverse object
    **/
    boolean forwardPressed = false;
    boolean missileFired = false;

    protected SimpleUniverse simpleU;
    protected BranchGroup rootBranchGroup;
    protected content content;
    //these are the transform3Ds that are used as temporary position holders
    private Transform3D oldTrans = new Transform3D();
    private Transform3D oldTrans2 = new Transform3D();
    private Transform3D oldTrans3 = new Transform3D();
    private Transform3D oldTrans4 = new Transform3D();
```

```
private Transform3D oldTrans5 = new Transform3D();
//for missile
private Transform3D oldTrans6 = new Transform3D();
//for backGround
private Transform3D oldTrans7 = new Transform3D();
//for 2nd background to make it look like its continuous
private Transform3D oldTrans8 = new Transform3D();

//these are the positions for all the objects
private Vector3d posValue = new Vector3d (0.0d,0.0d,-3.0d);
private Vector3d posValue2 = new Vector3d (0.0d,0.0d,-10.0d);
private Vector3d posValue3 = new Vector3d (5.0d,0.0d,5.0d);
private Vector3d posValue4 = new Vector3d (-10.0d,0.0d,-25.0d);
private Vector3d posValue5 = new Vector3d (20.0d,0.0d,-20.0d);
private Vector3d missilePosValue = new Vector3d (0.0d,0.0d,0.0d);
public Vector3d backGPosValue = new Vector3d (0.0d,0.0d,-43.0d);
//for 2nd background need to check the size if it
public Vector3d backGPosValue2= new Vector3d (360.0d,0.0d,-43.0d);

//these are the transform groups that hold the positions of each object
private Transform3D posTrans;
private Transform3D posTrans2;
private Transform3D posTrans3;
private Transform3D posTrans4;
private Transform3D posTrans5;
private Transform3D missileTrans;
private Transform3D backGTrans;
private Transform3D backGTrans2;
private Transform3D scaleTrans;

//these transform groups will be set to the transfom3D's that correspond
private TransformGroup mainTG = new TransformGroup();
private TransformGroup TG1 = new TransformGroup();
private TransformGroup TG2 = new TransformGroup();
private TransformGroup TG3 = new TransformGroup();
private TransformGroup TG4 = new TransformGroup();
private TransformGroup TG5 = new TransformGroup();
//for missile
private TransformGroup missileTG = new TransformGroup();
//for background
private TransformGroup backGTG = new TransformGroup();
//for 2nd backG
private TransformGroup backGTG2 = new TransformGroup();

//the scale variable makes sure the objects are normal size
private float scale = 1.0f;

//totation of the camera from the front
private double rot = 0;
```

```
//these variable allow the object to be rotated correctly
private double hyp;
private double angle;

private double hyp2;
private double angle2;

private double hyp3;
private double angle3;

private double hyp4;
private double angle4;

private double hyp5;
private double angle5;

//variables set up for game logic
private int boxesCollected = 0;
private int missiles = 3;

private OverlayCanvas canvas3D;

//for texturing
protected Texture tBackground;
protected Texture tMissile;
//public Box box;

//this allows a window to be set up
public BasicConstruct(String name)
{
   // The next line will construct the window and name it
   // with the given name
   super(name);
   initial_setup();

   //this set up the branch group with the child of the main transform group
   rootBranchGroup.addChild(mainTG);

   //this allows the branchgroup to be changed
   rootBranchGroup.setCapability((BranchGroup.ALLOW_CHILDREN_WRITE));
   definePrimitive();

}

protected void initial_setup()
{
   getContentPane().setLayout(new BorderLayout());
   GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();

   //canvas created for a world to be created
```

```
        canvas3D = new OverlayCanvas(config);

        getContentPane().add("Center",canvas3D);

        //the universe is being added to the canvas
        simpleU = new SimpleUniverse(canvas3D);

        rootBranchGroup = new BranchGroup();

        //command listener being added to the canvas to so controls can work
        canvas3D.addKeyListener(this);


    }

    public void addBox(float x, float y, float z, Color3f diffuse, Color3f specular)
    {
        //apearence needed for when creating the box
        Appearance app = new Appearance();
        Material mat = new Material();
        mat.setDiffuseColor(diffuse);
        mat.setSpecularColor(specular);
        mat.setShininess(5.0f);
        app.setMaterial(mat);

        //for background
        Color3f blackx = new Color3f(0.0f, 0.0f, 0.0f);

        Color3f whitex = new Color3f(1.0f, 1.0f, 1.0f);

        Color3f redx = new Color3f(0.7f, .75f, .75f);

        //loading up the texture and setting its attributes
        TextureLoader loadBackG = new
TextureLoader("360picture.png","LUMINENCE",new Container());
        tBackground = loadBackG.getTexture();
        tBackground.setBoundaryModeS(Texture.WRAP);
        tBackground.setBoundaryModeT(Texture.WRAP);
        tBackground.setBoundaryColor(new Color4f(0.0f,1.0f,0.0f,0.0f));

        TextureAttributes backGAtt = new TextureAttributes();
        backGAtt.setTextureMode(TextureAttributes.REPLACE);

        Appearance backGApp = new Appearance();
        backGApp.setTexture(tBackground);
        backGApp.setTextureAttributes(backGAtt);
        backGApp.setMaterial(new Material(redx,blackx,redx,blackx,1.0f));
        int primflagsx = Primitive.GENERATE_NORMALS +
Primitive.GENERATE_TEXTURE_COORDS;
```

```
//for missile so it has a different colour
Appearance app2 = new Appearance();
Material mat2 = new Material();
mat2.setDiffuseColor(new Color3f (0.0f,0.0f,1.0f));
mat2.setSpecularColor(new Color3f (1.0f,1.0f,1.0f));
mat2.setShininess(5.0f);
app2.setMaterial(mat2);

//setting up all the boxes within the world
Box box = new Box(x,y,z,app);
posTrans= new Transform3D();
posTrans.setTranslation(posValue);

//adding the boxes to the transform group and setting their position
TG1.addChild(box);//tg.addChild(box);
TG1.setTransform(posTrans);
//then add it to the rootBranchGroup
//rootBranchGroup.addChild(tg);
Box box2 = new Box(x,y,z,app);
posTrans2= new Transform3D();
posTrans2.setTranslation(posValue2);
TG2.addChild(box2);
TG2.setTransform(posTrans2);

Box box3 = new Box(x,y,z,app);
posTrans3= new Transform3D();
posTrans3.setTranslation(posValue3);
TG3.addChild(box3);
TG3.setTransform(posTrans3);

Box box4 = new Box(x,y,z,app);
posTrans4= new Transform3D();
posTrans4.setTranslation(posValue4);
TG4.addChild(box4);
TG4.setTransform(posTrans4);

Box box5 = new Box(x,y,z,app);
posTrans5= new Transform3D();
posTrans5.setTranslation(posValue5);
TG5.addChild(box5);
TG5.setTransform(posTrans5);

//missile being created wutg different variable as it is to be smaller
//and a different colour
Box missile = new Box(0.025f,0.025f,0.025f,app2);
missileTrans= new Transform3D();
missileTrans.setTranslation(missilePosValue);
missileTG.addChild(missile);
missileTG.setTransform(missileTrans);
```

```
        //bacckgrounds being set up to be large and have the texture attached
        Box backG = new Box(180.0f,23.0f,0.5f,primflagsx,backGApp);
        backGTrans = new Transform3D();
        backGTrans.setTranslation(backGPosValue);
        backGTG.addChild(backG);
        backGTG.setTransform(backGTrans);

        Box backG2 = new Box(180.0f,23.0f,0.5f,primflagsx,backGApp);
        backGTrans2 = new Transform3D();
        backGTrans2.setTranslation(backGPosValue2);
        backGTG2.addChild(backG2);
        backGTG2.setTransform(backGTrans2);

        //setting all the transform groups as children of the main one
        mainTG.addChild(TG1);
        mainTG.addChild(TG2);
        mainTG.addChild(TG3);
        mainTG.addChild(TG4);
        mainTG.addChild(TG5);
        mainTG.addChild(missileTG);
        mainTG.addChild(backGTG);
        mainTG.addChild(backGTG2);

    }

    private void definePrimitive(){
        //allows movement in each of the transform groups
        mainTG.setCapability(TransformGroup.ALLOW_TRANSFORM_READ);
        mainTG.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
        mainTG.setCapability(TransformGroup.ALLOW_CHILDREN_EXTEND);
        mainTG.setCapability(TransformGroup.ALLOW_CHILDREN_READ);
        mainTG.setCapability(TransformGroup.ALLOW_CHILDREN_WRITE);

        TG1.setCapability(TransformGroup.ALLOW_TRANSFORM_READ);
        TG1.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
        TG1.setCapability(TransformGroup.ALLOW_CHILDREN_EXTEND);
        TG1.setCapability(TransformGroup.ALLOW_CHILDREN_READ);
        TG1.setCapability(TransformGroup.ALLOW_CHILDREN_WRITE);

        TG2.setCapability(TransformGroup.ALLOW_TRANSFORM_READ);
        TG2.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
        TG2.setCapability(TransformGroup.ALLOW_CHILDREN_EXTEND);
        TG2.setCapability(TransformGroup.ALLOW_CHILDREN_READ);
        TG2.setCapability(TransformGroup.ALLOW_CHILDREN_WRITE);

        TG3.setCapability(TransformGroup.ALLOW_TRANSFORM_READ);
        TG3.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
        TG3.setCapability(TransformGroup.ALLOW_CHILDREN_EXTEND);
        TG3.setCapability(TransformGroup.ALLOW_CHILDREN_READ);
        TG3.setCapability(TransformGroup.ALLOW_CHILDREN_WRITE);
```

```
TG4.setCapability(TransformGroup.ALLOW_TRANSFORM_READ);
TG4.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
TG4.setCapability(TransformGroup.ALLOW_CHILDREN_EXTEND);
TG4.setCapability(TransformGroup.ALLOW_CHILDREN_READ);
TG4.setCapability(TransformGroup.ALLOW_CHILDREN_WRITE);

TG5.setCapability(TransformGroup.ALLOW_TRANSFORM_READ);
TG5.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
TG5.setCapability(TransformGroup.ALLOW_CHILDREN_EXTEND);
TG5.setCapability(TransformGroup.ALLOW_CHILDREN_READ);
TG5.setCapability(TransformGroup.ALLOW_CHILDREN_WRITE);

//for missile
missileTG.setCapability(TransformGroup.ALLOW_TRANSFORM_READ);
missileTG.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
missileTG.setCapability(TransformGroup.ALLOW_CHILDREN_EXTEND);
missileTG.setCapability(TransformGroup.ALLOW_CHILDREN_READ);
missileTG.setCapability(TransformGroup.ALLOW_CHILDREN_WRITE);

backGTG.setCapability(TransformGroup.ALLOW_TRANSFORM_READ);
backGTG.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
backGTG.setCapability(TransformGroup.ALLOW_CHILDREN_EXTEND);
backGTG.setCapability(TransformGroup.ALLOW_CHILDREN_READ);
backGTG.setCapability(TransformGroup.ALLOW_CHILDREN_WRITE);

backGTG2.setCapability(TransformGroup.ALLOW_TRANSFORM_READ);
backGTG2.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
backGTG2.setCapability(TransformGroup.ALLOW_CHILDREN_EXTEND);
backGTG2.setCapability(TransformGroup.ALLOW_CHILDREN_READ);
backGTG2.setCapability(TransformGroup.ALLOW_CHILDREN_WRITE);

rootBranchGroup.setCapability(BranchGroup.ALLOW_CHILDREN_WRITE);
//allow for detaching
rootBranchGroup.setCapability(BranchGroup.ALLOW_DETACH);



scaleTrans = new Transform3D();
scaleTrans.setScale(scale);

//setting the transform3Ds position values to the correct value
posTrans = new Transform3D();
posTrans.setTranslation(posValue);

posTrans2 = new Transform3D();
posTrans2.setTranslation(posValue2);

posTrans3 = new Transform3D();
posTrans3.setTranslation(posValue3);
```

```
        posTrans4 = new Transform3D();
        posTrans4.setTranslation(posValue4);

        posTrans5 = new Transform3D();
        posTrans5.setTranslation(posValue5);

        //for missile
        missileTrans = new Transform3D();
        missileTrans.setTranslation(missilePosValue);

        //for background
        backGTrans = new Transform3D();
        backGTrans.setTranslation(backGPosValue);

        backGTrans2 = new Transform3D();
        backGTrans2.setTranslation(backGPosValue2);

        //setting the transfrom groups positions
        TG1.setTransform(posTrans);
        TG2.setTransform(posTrans2);
        TG3.setTransform(posTrans3);
        TG4.setTransform(posTrans4);
        TG5.setTransform(posTrans5);
        //for missile
        missileTG.setTransform(missileTrans);
        //for background
        backGTG.setTransform(backGTrans);
        backGTG.setTransform(backGTrans2);

        //update sets all the positions
        update();

    }

    //this allows the canvas to be shown
    public void finalise()
    {
        simpleU.addBranchGraph(rootBranchGroup);
        simpleU.getViewingPlatform().setNominalViewingTransform();
    }

    private void setTransformations() {
        //setting all the transform groups according to the positions and scale
        //using a temporary transform3D and setting the proper transform
        //group to it for each object
        oldTrans.setIdentity();
        oldTrans.mul(posTrans);
        oldTrans.mul(scaleTrans);
        TG1.setTransform(oldTrans);
```

```
        oldTrans2.setIdentity();
        oldTrans2.mul(posTrans2);
        oldTrans2.mul(scaleTrans);
        TG2.setTransform(oldTrans2);


        oldTrans3.setIdentity();
        oldTrans3.mul(posTrans3);
        oldTrans3.mul(scaleTrans);
        TG3.setTransform(oldTrans3);


        oldTrans4.setIdentity();
        oldTrans4.mul(posTrans4);
        oldTrans4.mul(scaleTrans);
        TG4.setTransform(oldTrans4);


        oldTrans5.setIdentity();
        oldTrans5.mul(posTrans5);
        oldTrans5.mul(scaleTrans);
        TG5.setTransform(oldTrans5);


        oldTrans6.setIdentity();
        oldTrans6.mul(missileTrans);
        oldTrans6.mul(scaleTrans);
        missileTG.setTransform(oldTrans6);


        oldTrans7.setIdentity();
        oldTrans7.mul(backGTrans);
        oldTrans7.mul(scaleTrans);
        backGTG.setTransform(oldTrans7);


        oldTrans8.setIdentity();
        oldTrans8.mul(backGTrans2);
        oldTrans8.mul(scaleTrans);
        backGTG2.setTransform(oldTrans8);

    }

    public void update()
    {
        //updating all the positions
        setTransformations();
    }
```

```
//sets the position of each of the objects and sets their rotation to rot
//this then calls update to set the position
public void setPosition(double x, double y, double z, double rot)
{
   posValue.x = x;
   posValue.y = y;
   posValue.z = z;
   posTrans.setIdentity();
   posTrans.rotY(Math.toRadians(rot*-1));
   posTrans.setTranslation(posValue);

   update();

}

public void setPosition2(double x, double y, double z, double rot)
{
   posValue2.x = x;
   posValue2.y = y;
   posValue2.z = z;
   posTrans2.setIdentity();
   posTrans2.rotY(Math.toRadians(rot*-1));
   posTrans2.setTranslation(posValue2);

   update();

}

public void setPosition3(double x, double y, double z, double rot)
{
   posValue3.x = x;
   posValue3.y = y;
   posValue3.z = z;
   posTrans3.setIdentity();
   posTrans3.rotY(Math.toRadians(rot*-1));
   posTrans3.setTranslation(posValue3);

   update();

}

public void setPosition4(double x, double y, double z, double rot)
{
   posValue4.x = x;
   posValue4.y = y;
   posValue4.z = z;
   posTrans4.setIdentity();
   posTrans4.rotY(Math.toRadians(rot*-1));
   posTrans4.setTranslation(posValue4);
```

```
    update();

}

public void setPosition5(double x, double y, double z, double rot)
{
    posValue5.x = x;
    posValue5.y = y;
    posValue5.z = z;
    posTrans5.setIdentity();
    posTrans5.rotY(Math.toRadians(rot*-1));
    posTrans5.setTranslation(posValue5);

    update();

}

//rotation taken out because it is always wanting the missile to point in
//the same direction
public void setPositionMissile(double x, double y, double z, double rot)
{
    missilePosValue.x = x;
    missilePosValue.y = y;
    missilePosValue.z = z;
    missileTrans.setIdentity();
    //missileTrans.rotY(Math.toRadians(rot*-1));
    missileTrans.setTranslation(missilePosValue);

    update();

}


//rotation taken out as the background should alway point to the front
public void setPositionBackG(double x, double y, double z, double rot)
{
    backGPosValue.x = x;
    backGPosValue.y = y;
    backGPosValue.z = z;
    backGTrans.setIdentity();
    //missileTrans.rotY(Math.toRadians(rot*-1));
    backGTrans.setTranslation(backGPosValue);

    update();

}

public void setPositionBackG2(double x, double y, double z, double rot)
{
```

```
    backGPosValue2.x = x;
    backGPosValue2.y = y;
    backGPosValue2.z = z;
    backGTrans2.setIdentity();
    //missileTrans.rotY(Math.toRadians(rot*-1));
    backGTrans2.setTranslation(backGPosValue2);

    update();

  }

  //get position just returns the value of the position
  public Vector3d getPosition()
  {
    return posValue;
  }

  public Vector3d getPosition2()
  {
    return posValue2;
  }

  public Vector3d getPosition3()
  {
    return posValue3;
  }

  public Vector3d getPosition4()
  {
    return posValue4;
  }

  public Vector3d getPosition5()
  {
    return posValue5;
  }

  public Vector3d getPosMis()
  {
    return missilePosValue;
  }

  public Vector3d getPosBackG()
  {
    return backGPosValue;
  }

  public Vector3d getPosBackG2()
  {
    return backGPosValue2;
```

```
}

//this returns the value for the rotation
public double getRot()
{
   return rot;
}

//rotations check set the rotation and angle valueso of each object
public Vector3d rotateR()
{
   rot=rot+1.0;
   angle = angle+1.0;
   //using sin to get the new x value of the poition
   posValue.x = hyp * Math.sin(Math.toRadians(angle));
   //using cos to get the new z value of the poition
   posValue.z = hyp * Math.cos(Math.toRadians(angle));
   return posValue;
}

 public Vector3d rotateL()
{

   rot=rot-1.0;
   angle= angle-1.0;
   posValue.x = hyp * Math.sin(Math.toRadians(angle));
   posValue.z = hyp * Math.cos(Math.toRadians(angle));
   return posValue;
}

public Vector3d rotateR2()
{
   //rot=rot+1.0;
   angle2 = angle2+1.0;
   posValue2.x = hyp2 * Math.sin(Math.toRadians(angle2));
   posValue2.z = hyp2 * Math.cos(Math.toRadians(angle2));
   return posValue2;
}

 public Vector3d rotateL2()
{

   //rot=rot-1.0;
   angle2= angle2-1.0;
   posValue2.x = hyp2 * Math.sin(Math.toRadians(angle2));
   posValue2.z = hyp2 * Math.cos(Math.toRadians(angle2));
   return posValue2;
}

public Vector3d rotateR3()
```

```
{
   //rot=rot+1.0;
   angle3 = angle3+1.0;
   posValue3.x = hyp3 * Math.sin(Math.toRadians(angle3));
   posValue3.z = hyp3 * Math.cos(Math.toRadians(angle3));
   return posValue3;
}

 public Vector3d rotateL3()
{

   //rot=rot-1.0;
   angle3= angle3-1.0;
   posValue3.x = hyp3 * Math.sin(Math.toRadians(angle3));
   posValue3.z = hyp3 * Math.cos(Math.toRadians(angle3));
   return posValue3;
}

public Vector3d rotateR4()
{
   //rot=rot+1.0;
   angle4 = angle4+1.0;
   posValue4.x = hyp4 * Math.sin(Math.toRadians(angle4));
   posValue4.z = hyp4 * Math.cos(Math.toRadians(angle4));
   return posValue4;
}

 public Vector3d rotateL4()
{

   //rot=rot-1.0;
   angle4= angle4-1.0;
   posValue4.x = hyp4 * Math.sin(Math.toRadians(angle4));
   posValue4.z = hyp4 * Math.cos(Math.toRadians(angle4));
   return posValue4;
}

public Vector3d rotateR5()
{
   //rot=rot+1.0;
   angle5 = angle5+1.0;
   posValue5.x = hyp5 * Math.sin(Math.toRadians(angle5));
   posValue5.z = hyp5 * Math.cos(Math.toRadians(angle5));
   return posValue5;
}

 public Vector3d rotateL5()
{

   //rot=rot-1.0;
```

```
        angle5= angle5-1.0;
        posValue5.x = hyp5 * Math.sin(Math.toRadians(angle5));
        posValue5.z = hyp5 * Math.cos(Math.toRadians(angle5));
        return posValue5;
    }

    //finding out the hypotenuse of each object in the world using triginometry
    public double getHyp()
    {
        double temp = (getPosition().x*getPosition().x)+(getPosition().z*getPosition().z);
        hyp = Math.sqrt(temp);

        //making sure that when the hyp is -ve it still works
        if (getPosition().x < 0.0d /*|| getPosition().z < 0.0d*/)
            hyp = hyp*-1;
        return hyp;
    }

    //finding the using the hypotenuse found before using the inverse cosine
    public double getAngle()
    {
        double temp;
        temp = /*Math.toDegrees(*/Math.acos(getPosition().z/hyp)/*)*/;
        angle = Math.toDegrees(temp);
        //return angle;
        //printing out cos for testing
        System.out.print("cos of angle == "+getPosition().z/hyp);
        return angle;
    }

    public double getHyp2()
    {
        double                                          temp                                          =
(getPosition2().x*getPosition2().x)+(getPosition2().z*getPosition2().z);
        hyp2 = Math.sqrt(temp);

        //making sure that when the hyp is -ve it still works
        if (getPosition2().x < 0.0d /*|| getPosition().z < 0.0d*/)
            hyp2 = hyp2*-1;
        return hyp2;
    }

    public double getAngle2()
    {
        double temp;
        temp = /*Math.toDegrees(*/Math.acos(getPosition2().z/hyp2)/*)*/;
        angle2 = Math.toDegrees(temp);
        //return angle;
        System.out.print("cos of angle == "+getPosition2().z/hyp2);
        return angle2;
```

```
    }

    public double getHyp3()
    {
        double temp = (getPosition3().x*getPosition3().x)+(getPosition3().z*getPosition3().z);
        hyp3 = Math.sqrt(temp);

        //making sure that when the hyp is -ve it still works
        if (getPosition3().x < 0.0d /*|| getPosition().z < 0.0d*/)
            hyp3 = hyp3*-1;
        return hyp3;
    }

    public double getAngle3()
    {
        double temp;
        temp = /*Math.toDegrees(*/Math.acos(getPosition3().z/hyp3)/*)*/;
        angle3 = Math.toDegrees(temp);
        //return angle;
        System.out.print("cos of angle == "+getPosition3().z/hyp3);
        return angle3;
    }

    public double getHyp4()
    {
        double temp = (getPosition4().x*getPosition4().x)+(getPosition4().z*getPosition4().z);
        hyp4 = Math.sqrt(temp);

        //making sure that when the hyp is -ve it still works
        if (getPosition4().x < 0.0d /*|| getPosition().z < 0.0d*/)
            hyp4 = hyp4*-1;
        return hyp4;
    }

    public double getAngle4()
    {
        double temp;
        temp = /*Math.toDegrees(*/Math.acos(getPosition4().z/hyp4)/*)*/;
        angle4 = Math.toDegrees(temp);
        //return angle;
        System.out.print("cos of angle == "+getPosition4().z/hyp4);
        return angle4;
    }

    public double getHyp5()
    {
        double temp = (getPosition5().x*getPosition5().x)+(getPosition5().z*getPosition5().z);
```

```
        hyp5 = Math.sqrt(temp);

        //making sure that when the hyp is -ve it still works
        if (getPosition5().x < 0.0d /*|| getPosition().z < 0.0d*/)
            hyp5 = hyp5*-1;
        return hyp5;
    }

    public double getAngle5()
    {
        double temp;
        temp = /*Math.toDegrees(*/Math.acos(getPosition5().z/hyp5)/*)*/;
        angle5 = Math.toDegrees(temp);
        //return angle;
        System.out.print("cos of angle == "+getPosition5().z/hyp5);
        return angle5;
    }


    //this was another method to get position but was never used
    public Vector3d getLoc()
    {
        //getRot();
        posValue.x = (hyp * Math.sin(getRot()));
        posValue.z = (hyp * Math.cos(getRot()));
        return posValue;
    }

    public void checkCollisions()
    {

        //checking the collisions for picking up the objects
        if (posValue.x>-0.50d && posValue.x<0.50d && posValue.z>-
0.50d&&posValue.z<0.50f && posValue.y<50.0d)
        {
            //boxes collected to being added to
            boxesCollected++;
            //y position being set to 100 so no collision can take place
            posValue.y = posValue.y+100.0d;
            //player gets another missile when they collect a box
            missiles++;
        }

        if (posValue2.x>-0.50d && posValue2.x<0.50d && posValue2.z>-
0.50d&&posValue2.z<0.50f && posValue2.y<50.0d)
        {
            boxesCollected++;
            posValue2.y = posValue2.y+100.0d;
            missiles++;
        }
```

```
      if (posValue3.x>-1.0d && posValue3.x<1.0d && posValue3.z>-
1.0d&&posValue3.z<1.0f && posValue3.y<50.0d)
      {
         boxesCollected++;
         posValue3.y = posValue3.y+100.0d;
         missiles++;
      }

      if (posValue4.x>-0.50d && posValue4.x<0.50d && posValue4.z>-
0.50d&&posValue4.z<0.50f && posValue4.y<50.0d)
      {
         boxesCollected++;
         posValue4.y = posValue4.y+100.0d;
         missiles++;
      }

      if (posValue5.x>-0.5d && posValue5.x<0.5d && posValue5.z>-
0.5d&&posValue5.z<0.5f && posValue5.y<50.0d)
      {
         boxesCollected++;
         posValue5.y = posValue5.y+100.0d;
         missiles++;
      }

      /*if (posValue.x>missilePosValue.x-1.0d && posValue.x<missilePosValue.x+1.0d
       && posValue.z>missilePosValue.z-1.0d  && posValue.z<missilePosValue.z+1.0d
&&
       posValue.y <50.0d)
      {
         boxesCollected++;
         posValue.y = posValue.y+100.0d;
      }*/
   }

   public void checkMisColl()
   {
      //missile collision checks with boxes
      if ((posValue.x>missilePosValue.x-0.67d) &&
            (posValue.x<missilePosValue.x+0.67d) &&
            (posValue.z>missilePosValue.z-0.67d) &&
            (posValue.z<missilePosValue.z+0.67d) && posValue.y <50.0d)
      {
         //adding to the total boxes collected
         boxesCollected++;
         //again setting the box to 100 on y so no collision takes place
         posValue.y = posValue.y+100.0d;
         setPosition(getPosition().x, getPosition().y, getPosition().z, getRot());
         System.out.print("boxes Collected = "+boxesCollected);
         //missile fired set to false and back to origin
```

```
    missileFired = false;
    missilePosValue.z=0.0f;
    setPositionMissile(getPosMis().x, getPosMis().y, getPosMis().z, rot);
    //player looses a missile
    missiles--;

}

if ((posValue2.x>missilePosValue.x-0.67d) &&
        (posValue2.x<missilePosValue.x+0.67d) &&
        (posValue2.z>missilePosValue.z-0.67d) &&
        (posValue2.z<missilePosValue.z+0.67d) && posValue2.y <50.0d)
{
    boxesCollected++;
    posValue2.y = posValue2.y+100.0d;
    setPosition2(getPosition2().x, getPosition2().y, getPosition2().z, getRot());
    System.out.print("boxes Collected = "+boxesCollected);
    missileFired = false;
    missilePosValue.z=0.0f;
    setPositionMissile(getPosMis().x, getPosMis().y, getPosMis().z, rot);
    missiles--;
}

if ((posValue3.x>missilePosValue.x-0.67d) &&
        (posValue3.x<missilePosValue.x+0.67d) &&
        (posValue3.z>missilePosValue.z-0.67d) &&
        (posValue3.z<missilePosValue.z+0.67d) && posValue3.y <50.0d)
{
    boxesCollected++;
    posValue3.y = posValue3.y+100.0d;
    setPosition3(getPosition3().x, getPosition3().y, getPosition3().z, getRot());
    System.out.print("boxes Collected = "+boxesCollected);
    missileFired = false;
    missilePosValue.z=0.0f;
    setPositionMissile(getPosMis().x, getPosMis().y, getPosMis().z, rot);
    missiles--;
}


if ((posValue4.x>missilePosValue.x-0.67d) &&
        (posValue4.x<missilePosValue.x+0.67d) &&
        (posValue4.z>missilePosValue.z-0.67d) &&
        (posValue4.z<missilePosValue.z+0.67d) && posValue4.y <50.0d)
{
    boxesCollected++;
    posValue4.y = posValue4.y+100.0d;
    setPosition4(getPosition4().x, getPosition4().y, getPosition4().z, getRot());
    System.out.print("boxes Collected = "+boxesCollected);
    missileFired = false;
    missilePosValue.z=0.0f;
```

```
            setPositionMissile(getPosMis().x, getPosMis().y, getPosMis().z, rot);
            missiles--;
        }

        if ((posValue5.x>missilePosValue.x-0.67d) &&
                (posValue5.x<missilePosValue.x+0.67d) &&
                (posValue5.z>missilePosValue.z-0.67d) &&
                (posValue5.z<missilePosValue.z+0.67d) && posValue5.y <50.0d)
        {
            boxesCollected++;
            posValue5.y = posValue5.y+100.0d;
            setPosition5(getPosition5().x, getPosition5().y, getPosition5().z, getRot());
            System.out.print("\nboxes Collected = "+boxesCollected);
            missileFired = false;
            missilePosValue.z=0.0f;
            setPositionMissile(getPosMis().x, getPosMis().y, getPosMis().z, rot);
            missiles--;
        }

    }

    public void fireMissile()
    {
        //missile position being taken away on the z, to move the poition
        //forward into the screen
        missilePosValue.z = missilePosValue.z-0.0009d;
        setPositionMissile(getPosMis().x, getPosMis().y, getPosMis().z, 0.0f);
        //collision being checked everytime this is run
        checkMisColl();
    }

    //adding a light into the world
    public void addDirectionalLight(Vector3f direction, Color3f color)
    {
        //create bounding sphere for lights
        BoundingSphere bounds = new BoundingSphere();
        bounds.setRadius(100000000d);

        //then create a directional light with the given
        //direction and color
        DirectionalLight lightD = new DirectionalLight(color,direction);
        lightD.setInfluencingBounds(bounds);

        //then add it to the root branchGroup
        rootBranchGroup.addChild(lightD);
    }

    //taking 1 off the missiles
    public void missilesMinus()
    {
```

```
    missiles = missiles-1;
}

//returning how many boxes have been collected
public int getBoxesCollected()
{
    return boxesCollected;
}

//returns how many missiles are remaining
public int getMissiles()
{
    return missiles;
}

//returnung the canvas
public OverlayCanvas getCanvas()
{
    return canvas3D;
}

public void keyTyped(KeyEvent e) {
    System.out.print("\nkey typed");
}

public void keyPressed(KeyEvent e) {
    if (missileFired == false)
    {
        //A being pressed
        if (e.getKeyCode() == KeyEvent.VK_A){

            //checking for other keypresses
            if (forwardPressed == false)
            {
                //if not pressed it finds all the hypotenuses and angles
                getHyp();
                getHyp2();
                getHyp3();
                getHyp4();
                getHyp5();
                getAngle();
                getAngle2();
                getAngle3();
                getAngle4();
                getAngle5();
                forwardPressed = true;
            }

            //rotating each object then setting their positions
            rotateL();
```

```
            rotateL2();
            rotateL3();
            rotateL4();
            rotateL5();
            //background setting to plus 1 so the background scrolls giving
            //the impression of a real life background
            backGPosValue.x = backGPosValue.x+1.0f;
            backGPosValue2.x = backGPosValue2.x+1.0f;
            setPositionBackG(getPosBackG().x, getPosBackG().y, getPosBackG().z, rot);
            setPositionBackG2(getPosBackG2().x, getPosBackG2().y, getPosBackG2().z,
rot);
            /*rot=rot-1;*/
            //setPosition(rotateL().x, rotateL().y, rotateL().z, getRot());
            setPosition(getPosition().x/*-0.1f*/, getPosition().y, getPosition().z, getRot());
            setPosition2(getPosition2().x, getPosition2().y, getPosition2().z, getRot());
            setPosition3(getPosition3().x, getPosition3().y, getPosition3().z, getRot());
            setPosition4(getPosition4().x, getPosition4().y, getPosition4().z, getRot());
            setPosition5(getPosition5().x, getPosition5().y, getPosition5().z, getRot());
            System.out.println("\nx postion = "+getPosition().x+"now\nz position =
"+getPosition().z+"\nrotation"+rot);

        }
        //need to set a and d to be rotations
        //same as A pressed but negative
        else if (e.getKeyCode() == KeyEvent.VK_D)
        {
            if (forwardPressed == false)
            {
                getHyp();
                getHyp2();
                getHyp3();
                getHyp4();
                getHyp5();
                getAngle();
                getAngle2();
                getAngle3();
                getAngle4();
                getAngle5();
                forwardPressed = true;
            }
            //getHyp();
            //getAngle();
            rotateR();
            rotateR2();
            rotateR3();
            rotateR4();
            rotateR5();
            /*if (backGPosValue.x <-90.0f)
                backGPosValue.x = backGPosValue.x+180.0f;
            else if (backGPosValue.x >90.0f)
```

```
            backGPosValue.x = backGPosValue.x-180.0f;*/
            backGPosValue.x = backGPosValue.x-1.0f;
            backGPosValue2.x=backGPosValue2.x-1.0f;
            setPositionBackG(getPosBackG().x, getPosBackG().y, getPosBackG().z, rot);
            setPositionBackG2(getPosBackG2().x, getPosBackG2().y, getPosBackG2().z,
rot);
            /*posValue.x = posValue.x+0.1f;
            rot=rot+1;*/
            //setPosition(rotateR().x, rotateR().y, rotateR().z, getRot());
            setPosition(getPosition().x/*+0.1f*/, getPosition().y, getPosition().z,
getRot());
            setPosition2(getPosition2().x, getPosition2().y, getPosition2().z, getRot());
            setPosition3(getPosition3().x, getPosition3().y, getPosition3().z, getRot());
            setPosition4(getPosition4().x, getPosition4().y, getPosition4().z, getRot());
            setPosition5(getPosition5().x, getPosition5().y, getPosition5().z, getRot());
            System.out.println("\nx postion = "+getPosition().x+"now\nz position =
"+getPosition().z+"\nrotation"+rot);
        }

        //W pressed
        else if (e.getKeyCode() == KeyEvent.VK_W)
        {
            //setting the boolean to true to show the button has been
            //pressed
            forwardPressed = true;
            //hyp = hyp+0.1d;
           // getLoc();

            //all z positions being added to and set
            posValue.z = posValue.z+0.1f;//UNCOMMENT THIS
            //setPosition(getLoc().x, getLoc().y, getLoc().z, getRot());
            setPosition(getPosition().x, getPosition().y, getPosition().z, getRot());
            System.out.println("\nz postion = "+getPosition().z+ "now");
            //hypotenuses and angles being found
            getHyp();
            getAngle();
            //for 2nd object
            posValue2.z = posValue2.z+0.1f;
            setPosition2(getPosition2().x, getPosition2().y, getPosition2().z, rot);
            getHyp2();
            getAngle2();

            posValue3.z = posValue3.z+0.1f;
            setPosition3(getPosition3().x, getPosition3().y, getPosition3().z, rot);
            getHyp3();
            getAngle3();

            posValue4.z = posValue4.z+0.1f;
            setPosition4(getPosition4().x, getPosition4().y, getPosition4().z, rot);
            getHyp4();
```

```
            getAngle4();

            posValue5.z = posValue5.z+0.1f;
            setPosition5(getPosition5().x, getPosition5().y, getPosition5().z, rot);
            getHyp5();
            getAngle5();

            //background being moved towards camera however slower than
            //objects to stop it covering objects
            backGPosValue.z = backGPosValue.z+0.05f;
            backGPosValue2.z = backGPosValue2.z+0.05f;
            setPositionBackG(getPosBackG().x, getPosBackG().y, getPosBackG().z, rot);
            setPositionBackG2(getPosBackG2().x, getPosBackG2().y, getPosBackG2().z,
rot);

            //collisions checked at each time button is pressed
            checkCollisions();
        }

        //same as W but reversed
        else if (e.getKeyCode() == KeyEvent.VK_S)
        {
            forwardPressed=true;
            //hyp = hyp-0.1d;
            //getLoc();
            posValue.z = posValue.z-0.1f; //UNCOMMENT THIS
            //setPosition(rotateR().x, rotateR().y, rotateR().z, getRot());
            //for first object
            setPosition(getPosition().x, getPosition().y, getPosition().z, getRot());
            System.out.println("\nz postion = "+getPosition().z+" now");
            getHyp();
            getAngle();
            //for 2nd object
            posValue2.z = posValue2.z-0.1f;
            setPosition2(getPosition2().x, getPosition2().y, getPosition2().z, rot);
            getHyp2();
            getAngle2();

            posValue3.z = posValue3.z-0.1f;
            setPosition3(getPosition3().x, getPosition3().y, getPosition3().z, rot);
            getHyp3();
            getAngle3();

            posValue4.z = posValue4.z-0.1f;
            setPosition4(getPosition4().x, getPosition4().y, getPosition4().z, rot);
            getHyp4();
            getAngle4();

            posValue5.z = posValue5.z-0.1f;
            setPosition5(getPosition5().x, getPosition5().y, getPosition5().z, rot);
```

```
            getHyp5();
            getAngle5();

            //if (backGPosValue.z <43.0d)
            backGPosValue.z = backGPosValue.z-0.05f;
            backGPosValue2.z = backGPosValue2.z-0.05f;
            //else
                //backGPosValue.z = 44.0d;
            setPositionBackG(getPosBackG().x, getPosBackG().y, getPosBackG().z, rot);
            setPositionBackG2(getPosBackG2().x, getPosBackG2().y, getPosBackG2().z,
rot);
            checkCollisions();
        }

        //F pressed
        else if (e.getKeyCode() == KeyEvent.VK_F)
        {
            //checking the player has missiles left
            if (missiles >0)
                //missileFired set to true so the while loop can work
                missileFired = true;
        }
    }
  }

  public void keyReleased(KeyEvent e) {
    System.out.println("\nreleased key");

    //throw new UnsupportedOperationException("Not supported yet.");
  }
}
```

## Appendix C OverlayCanvas.java

```
package mariokartgame;

import java.awt.Color;
import java.awt.Graphics2D;
import java.awt.GraphicsConfiguration;
import javax.media.j3d.Canvas3D;
/**
 *
 * @author Yosemite Sam
 **/

//new canvas class which changed the post reneder method
class OverlayCanvas extends Canvas3D {

    //variables set so the post render can be used straight away
    protected String str = new String("a");
    int x = 10;
    int y = 10;


    public OverlayCanvas(GraphicsConfiguration gconfig/*, int w, int h*/)
    {
        //using the super constuctor
        super(gconfig);
        //getGraphics2D().setBackground(Color.BLUE);
    }

    //the variables can be set using this method
    public void setVariables(String string, int xx, int yy)
    {
        str = string;
        x = xx;
        y = yy;
    }

    //this is used to display text on the screen
    @Override
    public void postRender(/*String str, int x, int y*/)
    {


        //getGraphics2D().drawString(str, x, y);
        getGraphics2D().drawString(str, x, y);
        //repaint();
        getGraphics2D().flush(true);
        //postRender();
    }
```

```
}
```

## Appendix D Content.java

This was not actually used in the final program.

```
package mariokartgame;

import com.sun.j3d.utils.geometry.*;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import javax.media.j3d.*;
import javax.media.j3d.Transform3D;
import javax.media.j3d.TransformGroup;
import javax.vecmath.*;

/**
 *
 * @author Yosemite Sam
 */

//this was not actually used as could not get working
public class content extends BranchGroup /*implements KeyListener*/ {
   //some help from http://www.mail-archive.com/java3d-interest@java.sun.com/msg23020.html

   private double scaleValue = 1.0f;
   private Vector3f posValue = new Vector3f (0.0f,0.0f,0.0f);
   private Transform3D posTrans;
   private Transform3D scaleTrans;
   private Transform3D oldTrans = new Transform3D();
   private TransformGroup mainTG = new TransformGroup();
   private float scale = 0.2f;
   private float dim1 = 0.5f;
   private Vector3f base = new Vector3f();
   private boolean solid = false;
   private Color3f color = new Color3f(1.0f,1.0f,1.0f);
   private Appearance planetApp;
   private Sphere mySphere;
   Color3f black  = new Color3f(0.2f,  0.2f,  0.2f);

   public content()
   {
      this.addChild(mainTG);
      this.setCapability((BranchGroup.ALLOW_CHILDREN_WRITE));

      System.out.println("\n Cannot simply define blank sphere3d");

      setVariables(solid,color,base,scale,dim1);

      definePrimitive();
```

```
        drawSphere(solid,dim1);

        setPosition(base.x,base.y,base.z);
        setScale(scale);

    }

    private void setVariables(boolean _solid, Color3f _color, Vector3f _base, float _scale,
float _dim1) {
        solid = _solid;
        color = _color;
        base = _base;
        scale = _scale;
        dim1 = _dim1;
    }

    private void drawSphere(boolean solid, float radius) {
        if (!solid) {
            ColoringAttributes ca = new ColoringAttributes();
            ca.setColor(color);

            PolygonAttributes pa = new PolygonAttributes();
            pa.setPolygonMode (PolygonAttributes.POLYGON_LINE);

            Appearance ap = new Appearance();
            ap.setColoringAttributes (ca);
            ap.setPolygonAttributes (pa);

            mainTG.addChild(new Sphere(radius,ap));
        }
        else if (solid) {
            planetApp = new Appearance();
            planetApp.setCapability(Appearance.ALLOW_MATERIAL_WRITE);

planetApp.setCapability(Appearance.ALLOW_COLORING_ATTRIBUTES_WRITE);
            TransparencyAttributes _transAttrs;
            float _transparency = 0.0f;
            float _curtransparency = 0.0f;

            Transform3D _transTrans = new Transform3D();
            _transAttrs = new TransparencyAttributes(TransparencyAttributes.NONE,
            _transparency);
            // Allow the transparency value to be overwritten.
            _transAttrs.setCapability(TransparencyAttributes.ALLOW_VALUE_WRITE);
            // Allow the transparency mode to be overwritten.
            _transAttrs.setCapability(TransparencyAttributes.ALLOW_MODE_WRITE);
            planetApp.setTransparencyAttributes(_transAttrs);
```

```
            Material m = new Material(color,black,color,black,64.0f);
            planetApp.setMaterial(m);

             // Draw the sphere for the planet and add the texture to it
            mainTG.addChild(mySphere = new Sphere(radius,
            Sphere.GENERATE_TEXTURE_COORDS                                    |
Sphere.GENERATE_NORMALS,8, planetApp));
            mySphere.setCapability(Shape3D.ALLOW_APPEARANCE_WRITE);
            mySphere.setCapability(Shape3D.ALLOW_APPEARANCE_READ);
            mySphere.setCapability(Shape3D.ALLOW_GEOMETRY_WRITE);

        }
    }

    private void definePrimitive(){
        //allows movement
        mainTG.setCapability(TransformGroup.ALLOW_TRANSFORM_READ);
        mainTG.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
        mainTG.setCapability(TransformGroup.ALLOW_CHILDREN_EXTEND);
        mainTG.setCapability(TransformGroup.ALLOW_CHILDREN_READ);
        mainTG.setCapability(TransformGroup.ALLOW_CHILDREN_WRITE);
        this.setCapability(BranchGroup.ALLOW_CHILDREN_WRITE);
        //allow for detaching
        this.setCapability(BranchGroup.ALLOW_DETACH);

        //
        //  Multiply together the SCALING, TRANSLATION matrices
        //  to form one TRANSFORMATION matrix.
        //
        //   newMatrix = [S][T]
        //
        //      [S] = Scaling Matrix
        //      [T] = Translation Matrix
        //

        // SCALING transformation matrix

        scaleTrans = new Transform3D();
        scaleTrans.setScale(scale);

        //translation transformation matrix (combined with previous scaling matrix
        posTrans = new Transform3D();
        posTrans.setTranslation(posValue);

        //set the final transformation for 'this' transform group to be the composite
        mainTG.setTransform(posTrans);
        // UPDATE all of the transformations (multiply them together into one)
        // ...then set the Transform3D of this TransformGroup
        update(); //uncomment when update in
```

```
    }

    public void setSphereAppearance(Material newmat) {
        planetApp.setMaterial(newmat);
        mySphere.setAppearance(planetApp);
    }

    /**
    *  Scales the size of the model in all directions.
    *  @param newscale   scaling factor for the VRML object
    */
    public void setScale(double newscale) {
        scaleValue = newscale;
        scaleTrans.setIdentity();
        scaleTrans.setScale(scaleValue);

        update();//same as above
    }


   /**
    *  Moves the model to any 3D position.
    *  @param x     x position
    *  @param y     y position
    *  @param z     z position
    */
    public void setPosition( double x, double y, double z ) {
        posValue.x = (float)x;
        posValue.y = (float)y;
        posValue.z = (float)z;
        posTrans.setIdentity();
        posTrans.setTranslation(posValue);

        update();//same again
    }

    public Vector3f getPosition()
    {
        return posValue;
    }

    /**
    *  Sets the scaling, position and rotation transforms and
    *  set the combined Transform (this.setTransform) of 'this'
    TransformGroup.
    *
    */
    private void setTransformations() {
        // Multiply all the rotation, translation and scaling matrices together
        oldTrans.setIdentity();
```

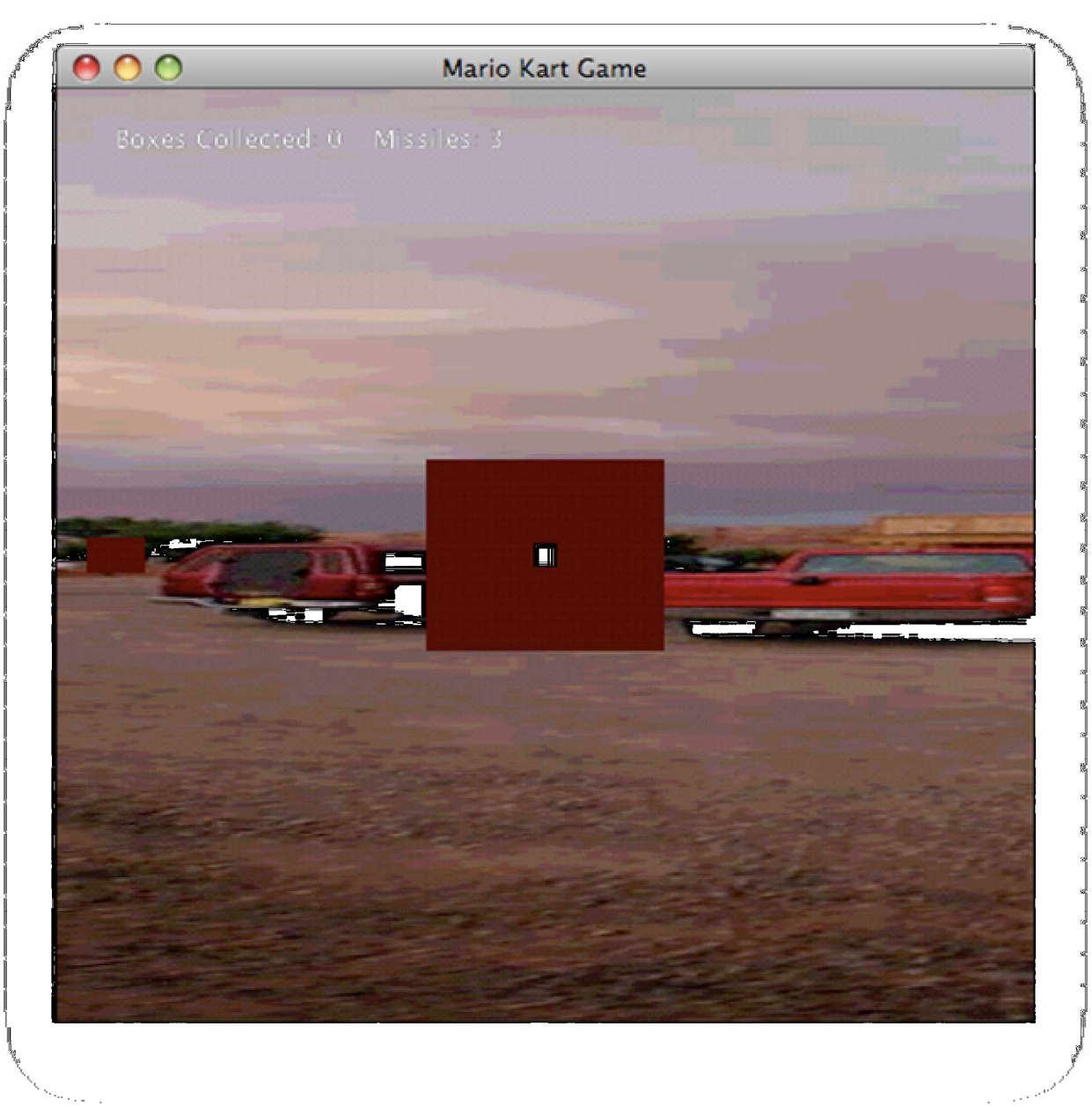
```
        // Multiply identity matrix by the TRANSLATION matrix
        oldTrans.mul(posTrans);
        //Multiply the combined TRANSLATION-ROTATION matrix by the SCALE matrix
        oldTrans.mul(scaleTrans);

        // set the TRANSFORM of this TransformGroup (TRANSLATION-ROTATION-SCALE)
        mainTG.setTransform(oldTrans);
    }


    /**
    *  Updates the transformations. This call is needed following any change
    in the planet.
    *
    *
    */

    private void update()
    {
        setTransformations();
        System.out.println("new x pos "+getPosition().x );
    }


    /*public void keyTyped(KeyEvent e) {
        throw new UnsupportedOperationException("Not supported yet.");
    }

    public void keyPressed(KeyEvent e) {
        if (e.getID() == KeyEvent.VK_LEFT)
        {
            setPosition(posValue.x+0.1f, posValue.y, posValue.z);
            System.out.println("new pos == "+posValue.x);
        }
        //throw new UnsupportedOperationException("Not supported yet.");
    }

    public void keyReleased(KeyEvent e) {
        throw new UnsupportedOperationException("Not supported yet.");
    }*/
}
```

## Appendix E Camera.java

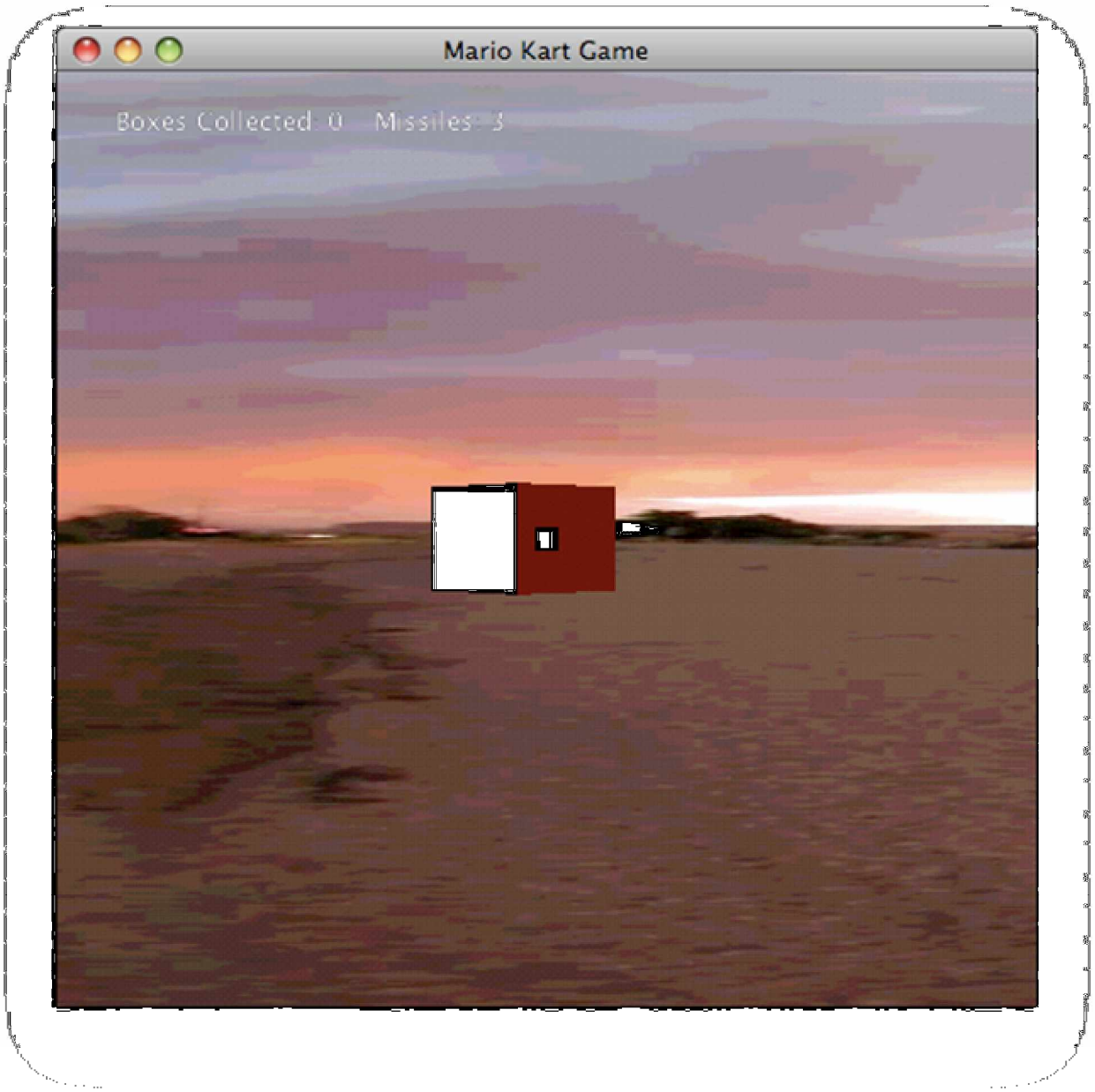
This class was not used in the actual final program.

```
package mariokartgame;

import com.sun.j3d.utils.behaviors.vp.ViewPlatformBehavior;
import com.sun.j3d.utils.universe.ViewingPlatform;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.util.Enumeration;
import javax.media.j3d.TransformGroup;
import javax.vecmath.Vector3f;

/**
 *
 * @author Yosemite Sam
 */

//class no used as it was unable to use the camera position
public class Camera /* extends ViewPlatformBehavior implements KeyListener*/ {
    Vector3f cameraPos;
    TransformGroup camTrans;
    //TransformGroup tg;

    public Camera(float x, float y, float z)
    {
        //Camera camera =new Camera(x,y,z);
        cameraPos = new Vector3f(x,y,z);
        //camTrans=new TransformGroup();
        //camTrans.addChild(camera);
    }

    public Vector3f getCameraPos()
    {
        return cameraPos;
    }

}
```

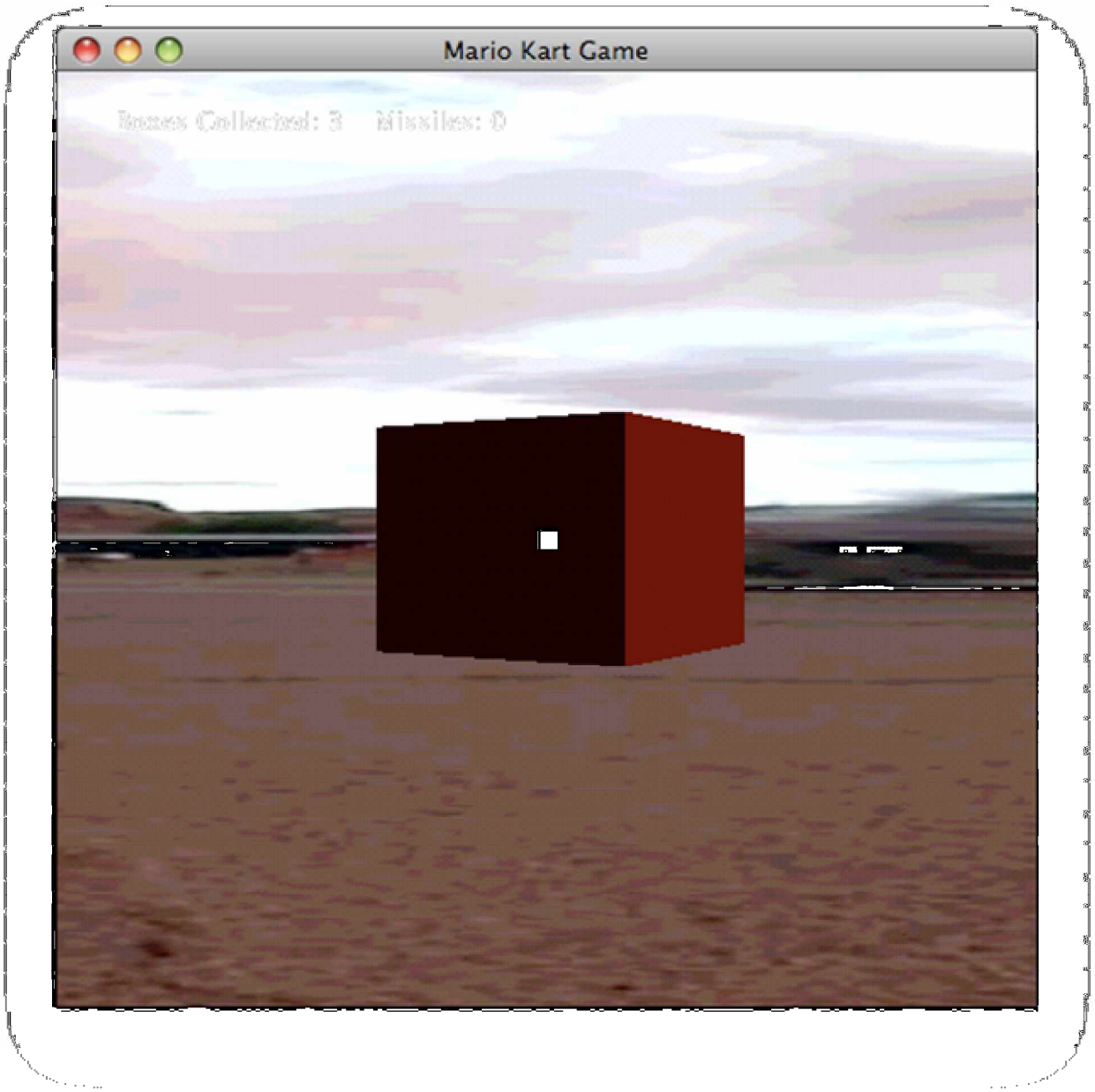
# Appendix F Screenshots



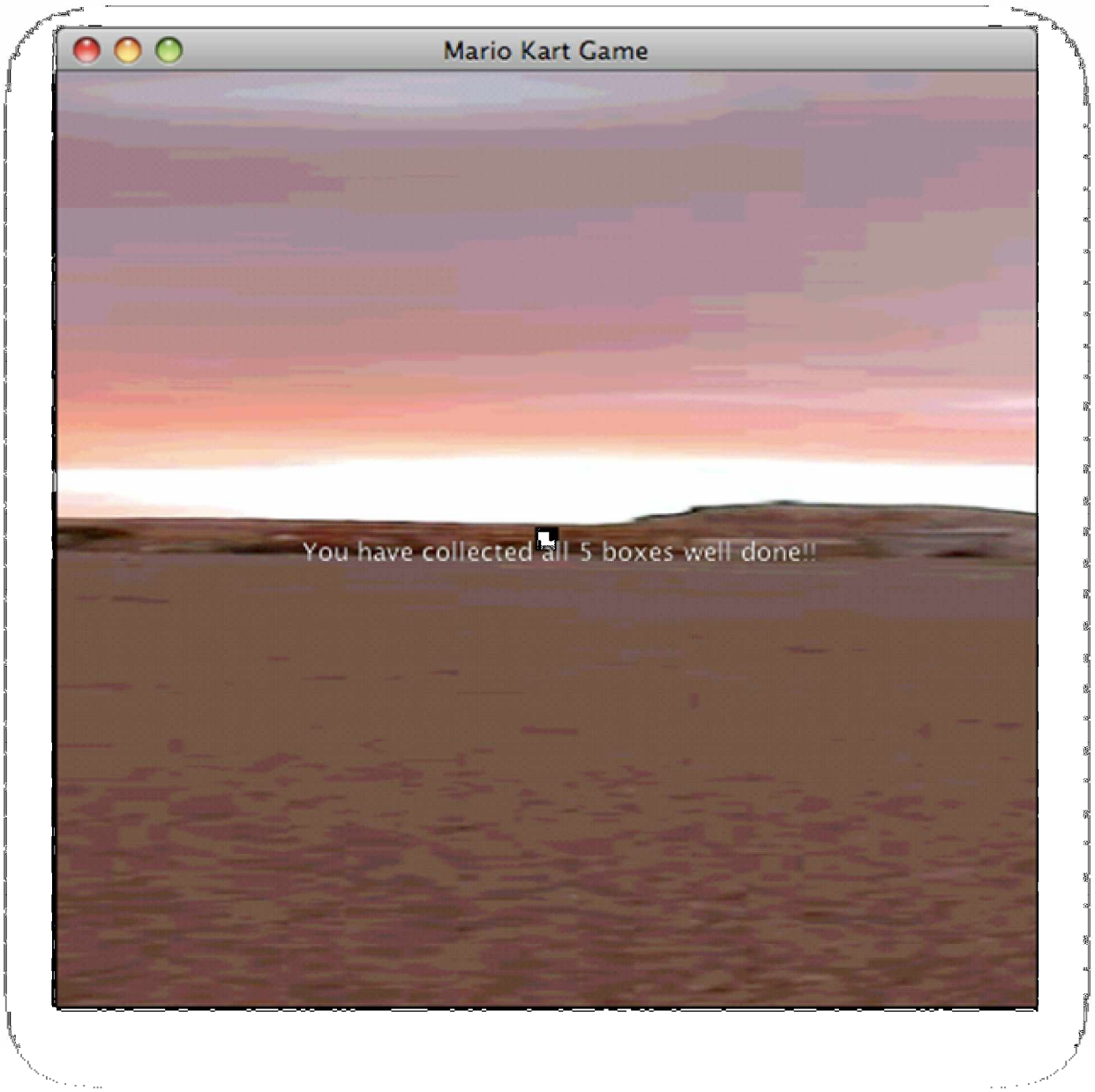

*Screen 1: Opening screen*




*Screen 2: Player moved from start*




*Screen 3: 3 boxes collected, 0 missiles left*




*Screen 4: Game finished with all boxes collected*