Electronics and Computer Science

Faculty of Physical Sciences and Engineering

University of Southampton

Imran Bepari

30/04/2018

Technical demonstration exploring movement methods in Virtual Reality games

Project Supervisor: David Millard

Second Examiner: Gary Wills

A project report submitted for the award of

MEng Computer Science

Abstract

When immersed in Virtual Reality (VR), the user is restricted to the physical limits of their room, and since games are usually larger than the confines of a small room area, different movement methods must be implemented to expand the virtual world for the user. However, it is not agreed on what the standard method for movement is. The aim of this project is to create a technical demonstration which explores different movement methods in VR.

The project will entail the creation of a small basic game that will feature three different movement types: teleportation, kinetic and vehicle-based, in an attempt to find a solution to this problem. Through research, testing and the collection of data, I will analyse the different levels of enjoyment that a user experiences with each movement method, and how appropriate they are for the game they have been created for.

Contents

1. Introduction

With recent release of Virtual Reality (VR) head mounted displays on a consumer level, the gaming industry has still yet to properly grasp how a VR game should be made on the scale of a AAA title, and how movement should work in it. In this context, AAA title is referred to as a game developed by mid-sized to large companies, with high development and marketing budgets, similar to the term "blockbuster" in the film industry [1].

Movement is a difficult subject in VR because of its boundaries. A user is restricted to the room they are in, so exploring a large level or world would require additional movement systems to make the game playable.

Many games simply fit into the users play space, not move them around or use a mechanic such as teleporting to move the player. While these solutions work for some genres, such as short independent games, a solution has not been found for large scale games with huge worlds, such as role-playing games and shooters.

For larger games, a teleport mechanic is an unsatisfactory solution, and doesn't work for the intended user experience. Multiplayer games are broken if a player can teleport around since players need to be visible at all times to be able to be hit by other players. Furthermore, teleporting can also be thematically unfitting for a game, for example, if a game has a medieval theme, players should not be able to move in a teleporting fashion.

In addition to this, motion sickness is the paramount problem when implementing different movement mechanics in the virtual world, and implementing a method that doesn't call on this effect can be difficult. Motion sickness is the significant reason that a true solution, for movement in VR, has been difficult to find, as different users have different levels of tolerance to the VR world.

To address this, I will create a VR shooter game as an attempt to research potential solutions to the problems I have mentioned. It will be a basic shooter game that will be played in a head mounted display, the HTC Vive, and utilize multiple different options for movement methods.

There will be 3 main movement mechanics:

a) Teleportation Movement
b) Kinetic Movement
c) Vehicle-Based Movement

Using user-based evaluation, I can obtain feedback for what the players think about each of the movement methods. This data will measure users' enjoyment and how the game makes them feel, allowing me to evaluate the effectiveness of each movement method for the game at hand.

2. Background Reading

This section covers the literature I have reviewed and the background research I have done in preparation for this project.

2.1. General Games Design

To consider creating and explaining a solution to a problem in the design of video games, the basics of video games design must be used as a foundation.

First and foremost, the basics of game design require a game to "advocate for the player" [2]. It's common for developers to add features to a game without realising that they may do nothing for the player, or even make a game worse for the player.

Game design is fundamentally difficult, as Schell, writer of "The Art of Game Design", lists multiple skills and requirements in the creation of a successful game, including engineering, mathematics and management. Mainly, being able to "dissect an experience" will be useful when developing in VR. He suggests that being able to tell why something's bad is a skill required for design, rather than just being able to tell that something is bad [3].

Unlike typical software, users interact with games very differently. The concept of Human-Computer Interaction (HCI) is a large focus in games design. If a game "is not compelling and entertaining, the product fails in the marketplace" [4]. It's been shown that HCI and games design can learn a lot from each other and are very similar studies. It can be assumed that when playing a game with the VR medium, it must be properly designed to be enjoyable.

A difference between typical software and games is the difference in control flow. Games will run code every "frame", which is a snapshot of the games current state, often at 60fps (frames per second) or even 90fps for VR. This means that the code I create will have to be optimal for performance to be upheld, since the engine will have to compute all the necessary processes in my game and draw to the headset every frame. Performance is important for reasons mentioned in Section 2.3.

In the writing of the Games Design document in this project (see Section 3.1), the core games development values were learnt about to aid in making a satisfactory solution.

2.2. Virtual Reality

Virtual Reality is the experience of another three-dimensional environment, usually through a head mounted display. The content that can be played through VR include images, videos and video games. VR offers these in a more immersive format, allowing users to believe they are truly in a picture, video or world by placing the users in them. Above all, interactivity included in VR is unlike typical media or even some traditional video games, as the use of motion controllers can allow for truly vivid and what feels to be "real" experiences [5].

This is achieved by giving each eye its own display with corrective lenses to emulate the way humans see in the real world. The displays will render at a slightly different angle to each other to achieve the 3D effect. Optionally, motion controllers can also be included to give the user a way to interact with the virtual world they are presented with.

As of the writing of this report, the hardware, consumer grade VR head mounted displays come in two distinct categories: Standing and Room Scale. It's important to know what hardware to be developing for along with it's capabilities, along with what sets them apart.

The most common and cheapest form of VR is through Standing, with headsets such as Google Cardboard costing as little as £11.55 [6], which utilizes the user's smartphone. Other Standing VR experiences include the Samsung Gear and the original Oculus Rift Development Kit 1. These head mounted displays only track the rotation of a user's head, rather than the position and rotation. These pose less interactivity than other head mounted displays, as only being able to turn the head restricts the movement of the user.

Room Scale is far more expensive and less common in the consumer world, simply because of the costs associated with acquiring one. Headsets in this category such as the HTC Vive and Oculus Rift Consumer Release exceed £399 [7][8] in price, in addition to requiring a computer powerful enough to accommodate it. These head mounted displays feature full positional and rotational tracking for the user, over a room, allowing for a more immersive experience.

As of now, most VR experiences running on Windows computers will interface through SteamVR, supported by OpenVR. OpenVR is an open source development kit that allows head mounted displays from any manufacturer to interact with software such as Steam, or in this case a game engine [9].

2.3. Motion/Cyber Sickness

Motion sickness induced by VR, or cyber sickness, is a large phenomenon in need of consideration when creating a game. It manifests as symptoms of "discomfort, eye strain, nausea etc" [10]. This can be caused by movement not caused by the user, similarly to a user getting motion sick in a moving vehicle. The difference between cyber sickness and actual motion sickness is that it's caused by the "visual perception of self-motion", rather than actual real motion [10].

2.3.1. Causes

VR requires high end hardware to meet the requirements of the high frame rates and resolution associated with it. Without meeting these requirements, VR experiences can be sub-optimal and even induce cyber sickness.

When applied to the movement methods mentioned below, many players experience this sickness when attempting to play VR games, especially large-scale ones that require the use of these movement methods to progress.

Rolnick and Lubow suggested that the brain not feeling in control of movement and being reactive in the perception of motion could be a cause of motion sickness [11]. The act of only watching movement can cause motion sickness, while proactively causing the motion rarely induces it.

The Field of View (FOV) has been observed to have an effect on whether a user experiences sickness, where while a wider FOV immerses the user more, it makes them more prone to sickness [12]. Most consumer head mounted displays, including the HTC Vive, Samsung Gear VR and the Oculus Rift include a 110º field of view [13].

2.3.2. Solutions

Suggested solutions for reducing sickness includes introducing a static frame of reference [14], where by adding physical context to a player's position, their brain can handle the movement they aren't experiencing. With light of this study, one of my implementations will combine the Smooth movement method (mentioned in Section 2.4.1), with a context addition to hopefully reduce how sick players feel.

As per mentioned in Section 2.3.1, if feeling not in control, one is more likely to feel motion sick. In order to give the player the most feeling of being in control, a solution could be to proactively have the user do actions associated with their movement, as mentioned by Rolnick and Lubow [11].

2.4. Examples of Movement in Virtual Reality

As the amount of literature on modern movement in VR is lacklustre, I turned to games developers and examples of games themselves in researching different types and implementations, along with their views and input for the topic.

VRemedyLabs talks about the approach of movement in VR, and how it can be remedied. They believe there isn't just a "single solution for locomotion in VR, or for the nausea" and that solutions have to be fine-tuned to a game [15]. It was indicated that there was a lack of immersive vast environments, with laments that they "didn't feel like a powerful hero immersed in a colossal world; I felt like a bound prisoner trapped in a tiny cell." [16] Kim Voll at Games Development Conference spoke about how human brains are "designed to move us through the world, not have the world move around us" [17], and how the hurdle of movement in VR is a biological one too. This alludes that above all, more trickery of the brain is required to create a good solution to movement in VR.

Steam is one of the biggest digital distribution platforms on the internet, and is considered to have 75% of that market space, as of October 2013 [18], making resourceful for analysing the success of video games. The Top Selling VR games

on Steam, as of 08/12/2017, were analysed for their methods for implementing movement, if any.

Game Name	Smooth Movement	Teleport Movement	Other
Raw Data	✓	✓	
Fallout 4 VR	✓	✓	
Doom VFR		✓	
Onward	✓		
Pavlov VR	✓		
Rick and Morty: Virtual Rick-ality		✓	
GORN			✓
Talos Principle VR		✓	✓
Arizona Sunshine	✓	✓	

Table 1: List of Top Selling Games on Steam for 08/12/2017.

2.4.1. Smooth Movement

These games employ smooth movement, where pushing a button would project the player in a direction, at a constant speed. It's most comparable to conventional game controls, since it acts like a control stick on a game controller. These games have a bigger action focus, and mostly have gun mechanics involved with them. As mentioned before, it can be predicted that this method is used, despite the cyber sickness it causes some people, because of the players fundamental need to be able to be hit by enemies. In addition to this, three of the games in the table are also multiplayer shooter games.

2.4.2. Teleport Movement

The considered default method of movement in VR is teleportation, due to how common it is in games. This movement involves a player selecting where they want to move in the virtual world, by pointing to it, and the game will teleport them there without any illusion of movement or motion. This is the safest way to move a player about a game without inducing cyber sickness, and as a result, many of the games in here appear to overlap with other categories, by using this method as a backup option. Raw Data is a prime example of a VR game that uses both smooth movement and teleportation, where each of them is optional and you can use whichever you want, whenever.

2.4.3. Other Movement – Kinetic

This category contains outlier movement methods within the table. Some in the selected list do movement significantly different to the other games. GORN uses a method where players' physical motion of their hands is translated into movement in game. This can be compared to pulling on a rope, to pull yourself forward. Talos Principle VR has a control where the game will allow you to

rotate yourself 90º on the spot, so the user doesn't have to reorient themselves in real life.

These analyses have influenced the design of my technical demonstration significantly, as my game will try to emulate three of these movement methods mentioned while making improvements to them.

2.5. Hardware Limitations in Virtual Reality

Hardware limitations are currently some of the most problematic limitations for the development of Virtual Reality, especially in the field of video games. Naturally, VR is expensive in terms of processing required, since the head mounted display must render and display two slightly different images, at a relatively high resolution and refresh rate. This section of background reading is important for learning to build and program for the VR medium.

To create the 3D effect of Virtual Reality, the computer must generate two images and different angles, equal to the distance between the user's eyes. This allows the users to perceive depth in the image when each image is being fed to each eye. Rendering an entire scene twice is extremely taxing on hardware, which is why games tend to have fewer objects in each scene.

Without a high resolution, users experience the "Screen Door effect" where users are able to distinguish between the different pixels and lines in the display, which can break immersion very quickly. [19] In addition to having the hardware do more work to solve this problem, the prices of displays that are head mounted and render at higher resolutions are extremely expensive and are not viable for the consumer market.

Figure 1: Image of the screen door effect, unknown source linked from [20]

Most VR hardware aims for a refresh rate of 90fps. Most games on typical video game consoles display at 30fps, however the optimum for typical viewing is around 60fps. 90fps is much higher than what typical hardware is used to exhibiting, especially twice at higher resolutions, but is a minimum requirement

in the VR medium. Human eyes are sensitive to frame rate changes [21], and an inconsistent frame rate can cause cyber/motion sickness, as mentioned in Section 2.3.

Missing any of the mentioned above requirements: frame rate, resolution and 3D effect, fundamentally breaks the experience, and as a result, most games have low graphical fidelity.

3. Design

This section contains the details of the design for the technical demonstration. Mainly, it'll describe how the game will work and what will be contained, along with requirements and details for the application itself.

3.1. Game Design

Full details of how the game will be designed can be found in the Game Design document, in Appendix A.

The Game Design document is a descriptive document of the design of a video game, which is continually updated and edited as development goes on [22]. It's used as a guide containing consolidated information of how a game is going to be and is useful in ensuring the game doesn't lose sight of its direction during development. When using a Game Design document, the game is subject to design changes as development goes on, as occasionally there are instances where a chosen implementation of a feature works better on paper than practically.

By trying to find out how different implementations of the solution work and feel, all of them need to be implemented to begin with. As mentioned before, there will be three main movement mechanics:

a) **Teleportation movement.** Pushing the main Face button on the Vive controller will allow the player to teleport to a destination of their choice, up to a certain distance away. Mainly used as the baseline to be compared against.
b) **Kinetic movement.** The movement of the character will be taken from the physical movement of the player. Pushing the Face button and making a walking motion with the controller will cause the player to move.
c) **Vehicle-Based movement.** Adding the context of a vehicle that the player is in may reduce the motion sickness of smooth movement. Free movement by simply touching on the Face button the direction the player wants to go.

These movement choices have been inspired by the background reading. The game will require use of the movement methods to finish the game, and the user will use all three movement types for the level, in separate playthroughs, before providing feedback for analysis.

The game will consist of a single level that will be played by the user, where the goal is to simply reach the end of the level, denoted by a flag. The level design will be centred around encouraging the player to move around and use the developed tools. In terms of structure, the level will be linear with a clear path to take, and the player will have to move to reach the goal.

The nature of the game will be a 'shooter' game. The user will encounter red enemies which they must shoot to defeat, and the enemies will be trying to shoot them back. As Section 2.4 mentions, since solutions should be fined tuned for the game, they will be built for combating these enemies.

For player health and lives, the player will die and restart the level when they sustain a certain amount of hits from the enemies. Health can be recovered using health packs found around the maps but must be reached and grabbed first.

These game design choices are made to encourage the users to use the movement options given to them, in order to complete the game. By forcing a user to adapt to a given controller scheme, they will be able to give real feedback on how the movement felt in context of helping them reach the goals in game.

3.2. Requirements

Table 2 below describes the functional and non-functional requirements of the proposed solutions. Above all, FR1-3 are compulsory requirements, which are bare minimum to initiate user-based testing. With these requirements implemented, I will be able to compare the different movement methods with the aid of user data.

Mainly, the game must run smoothly and utilize the VR medium properly, or the comparison won't be useful. Noticeable lag, as mentioned in Section 2.3.1, will cause more motion sickness than necessary which affects the results of the solutions.

ID	Requirement Description	Priority
FR1	Run on the HTC Vive	Compulsory
FR2	Make use of motion controllers	Compulsory
FR3	Feature at least two movement methods	Compulsory
FR4	Feature a third movement system	Optional
FR5	Have a basic game featuring the movement methods	Compulsory
NFR1	Run smoothly, with no noticeable lag	Compulsory
NFR2	Run on other Virtual Reality hardware	Optional
NFR3	Have multiple levels to test with	Optional

Table 2: Requirements table

3.3. UML Diagram

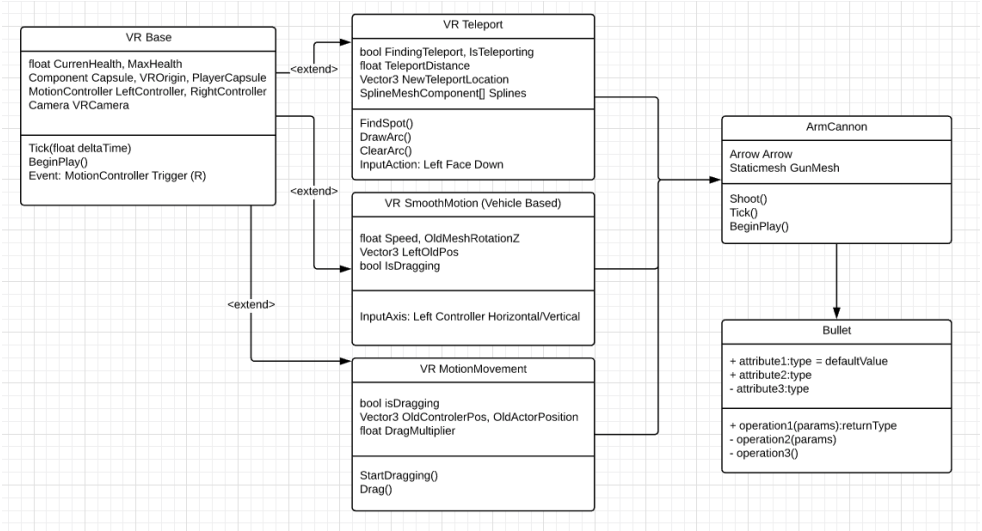


Figure 2: This UML diagram represents the basic structure of the player related classes.

The UML diagram, from Figure 2, is a simplified representation of the class layout for the game. The full size can be found in Appendix B. In addition to this, there is a separate UML diagram, Figure 3, describing the class layout for other mechanics within the game. These are largely disjointed from the player programmatically, yet still interact with them in game.

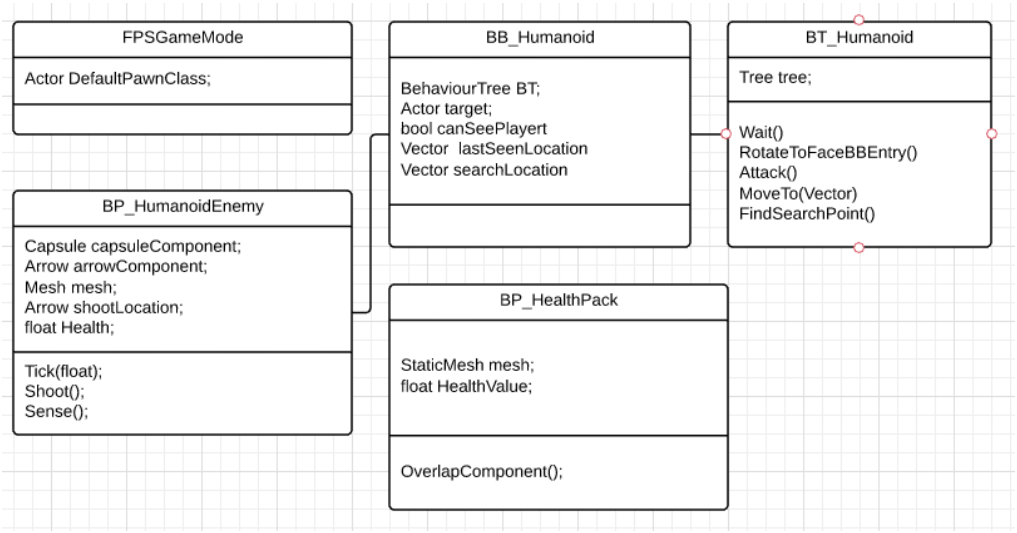


Figure 3: This UML diagram represents the other related classes needed inside the game.

The base of the VR player controller is the parent class to the different solutions, which makes implementing the individual solutions easier. The VR Base contains all the necessary components and variables to have the player exist in the game world, even if unable to do anything in the world. It contains both of the motion controllers included with VR headsets and children components. The parent also interfaces with SteamVR to work with the game engine, which is Unreal Engine 4 (UE4).

UE4 was chosen for reasons stated in Section 4.4. The main reason this chapter of this document is sparse in comparison to others is due to the fact that UE4 handles a lot of the foundations and bases for my application. This means I do not have to consider concepts such as MVC or modelling the application to fit SteamVR, since UE4 does it already. It allows me to solely focus on the VR movement methods I'm trying to create.

4. Implementation

This section will focus on the implementation of the demonstration. As mentioned before, this project was created using Unreal Engine 4, using it's visual C++ Blueprint methods. The VR with the HTC Vive works in Unreal Engine 4 through the SteamVR wrapper. The implementation contains many features to make the game complete enough to be playable, but for the purpose of conciseness, only significant and relevant features will be described below.

4.1. Movement Methods

The main work of this project was in the implementations of the movement methods, which are detailed below.

4.1.1. Teleport Movement

My rendition of a Teleportation Movement system has the user use the Face button on the Left Controller of the Vive. Pushing the button initiates teleportation, where it must be held to decide where the player will move to. Upon release, the player will be teleported. When the player holds the button down, the Vive controller will project a virtual arc, which will end at the destination where the user will be teleported. If the teleport location is invalid, then no arc will be shown.

Figure 4: Image of teleportation arc used to show where the player will be teleported

This is implemented with the use of predictive mechanics algorithms and a Navigational Mesh, where a Navigational Mesh is a method for representing the game world using polygons [23]. Usually, a Navigational Mesh is used in Artificial Intelligence (AI) to determine where an AI can or cannot traverse, however, in this case it's used to let the game know where the player can and

cannot teleport. Using this, I was able to make the teleportation so that players could not teleport to unintended places, through unintended means. These include teleporting through a ceiling or up a wall.

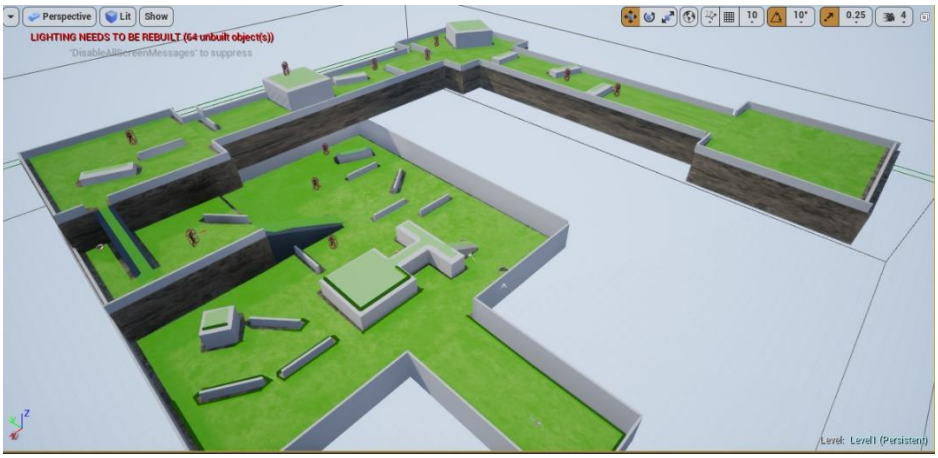


Figure 5: Image of the Navigational Mesh, as the visible green layer over the levels geometry

Using an arc to inform the player of their teleportation destination is a common occurrence in VR games. In order to draw the arc, a projectile launch is simulated from the tip of the controller, and the predicted path is drawn as the arc. To change the distance the player can teleport, I changed the speed and/or physics of the simulated projectile. This method of choosing the teleport destination allows the player to traverse vertically as well as horizontally, fairly.

4.1.2. Kinetic Movement

Kinetic movement features forcing the user to create movement with their body to induce movement in the virtual world. When the user pushed the Face button of either controller, they initiate movement. While the Face button is held, the user will move in the opposite direction of their arm movement. By swinging their arms in a walking motion and rhythmically pushing the Face button to pull at appropriate moments in their arms arc, the user can propel themselves forward.

This simply works by storing where the player has their hand when they push the button and move their global location relative to the movement of their hands movement. The hand positions exist relatively to the room-scale and headset locations in Unreal Engine 4, so the positions had to be converted to world space and reconverted to local space when calculating the distance to be moved.

Since the perception of motion where there is none seems to cause motion sickness (as mentioned in Section 2.3), this method aims to remedy motion sickness by using the motion induced by the hands as a placebo. In addition to this, giving the player more control and precision over where they move and how fast they move can remedy the sickness, while being a viable movement method in the context of a game.

The kinetic movement method was a challenge to implement, as it required more in-depth knowledge of the engine and its documentation to create. These problems are mentioned more in Section 4.6.

4.1.3. Vehicle-Based Movement

The Vehicle-Based movement was the most difficult to implement. It attempts to use the smooth motion movement mentioned before, without any of the negative side effects. This has been done by placing the user in a “mechanical suit”/ “robot” or whatever object is required. By creating this static frame of reference, the users brain is tricked into thinking nothing’s wrong when the body is being moved. The outer object is carrying the user, therefore there’s nothing wrong when movement is taking place.

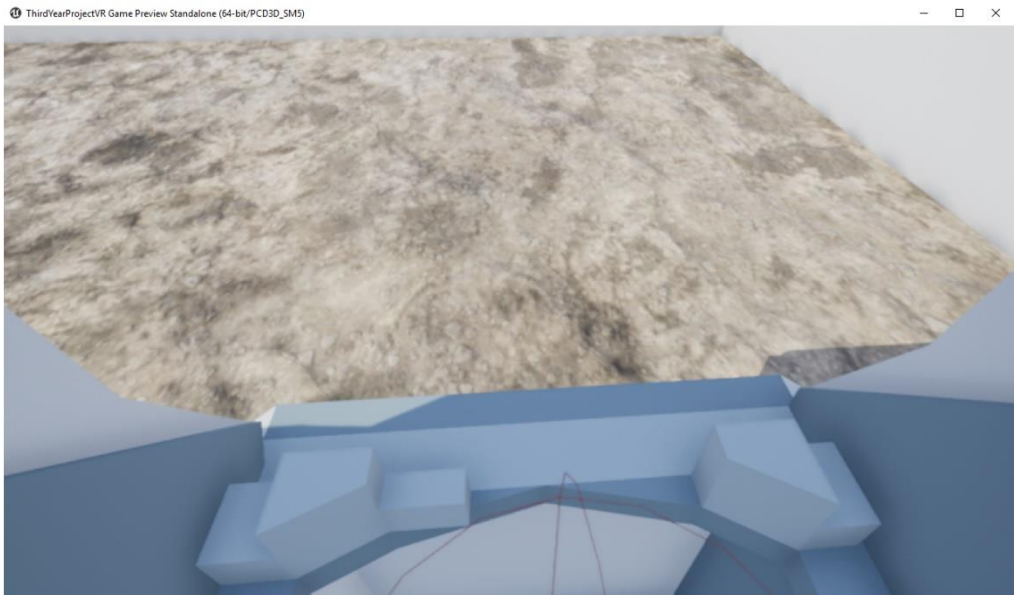


Figure 6: The view from inside the vehicle, within VR

The concept and reasoning behind this implementation can be found largely in Section 2.3, where I have attempted to create a solution using the research gathered. As mentioned, a static frame of reference is created to help reduce motion sickness, but in addition to this it works as an artificial means of reducing field of view for the player. By encapsulating the player inside the vehicle and limiting the players viewpoint via the window, the player has effectively a lower FOV, more akin to 70º to 90º rather than the typical 110º.

The player must control the vehicles orientation using the controller. By holding the trigger and orbiting the controller relative the players position, the vehicle will also change direction, altering the view point.

4.2. Enemy Artificial Intelligence

For the technical demonstration to work, I needed to put the player in a position where they were required to use the movement methods.

I implemented a basic humanoid enemy, where it can see the player, attempt to fire at them, and chase the player if they lose sight of them. This was done using behaviour trees in Unreal Engine 4. By having this AI proactively shoot the player, the player would be encouraged to move to evade the advancing AI enemy. This in turn naturally gets them to use the movement methods I've implemented and allows the player to truly feel the movement method. Evaluating different movement methods becomes easier when a player feels like they are more or less capable of escaping from the same type of enemy.

Behaviour trees are an easy way to implement basic AI functionality, for a purpose such as this. They were inherently built into Unreal Engine 4, making it easy to utilize them. Behaviour Trees allow a hierarchical way of organising tasks in a descending order, making the tree more scalable and usable[24]. I was easily able to create the behaviour of idling and waiting to see the player, followed by chasing and shooting the player successfully.

As mentioned in Section 4.1, Navigational Meshes were used to let the AI know where it could traverse. By using the same Navigational Mesh as the one in Section 4.1, I could ensure that the player and the AI could traverse the same parts of the level, which meant that the player could not exploit or be exploited by the AI and the Navigational Mesh.

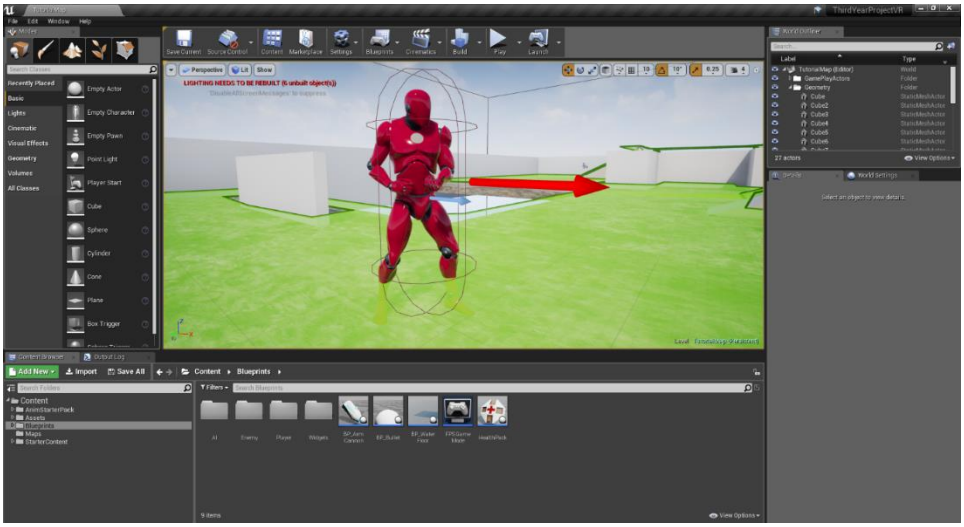


Figure 7: Image of the AI the player faces in the technical demonstration.

4.3. Gun and Projectile

Both the player and the enemies use the same projectile to deal damage to each other. The projectile is simply a default sphere from UE4 that is created at the tip of the players gun and launched in the direction that the gun is facing. The gun is implemented to be a child of the players class, where the player "possesses" a gun that's statically mounted to their arm. The player makes calls to gun object to fire, making it a modular attachment for both the player and the enemies. The orientation of the arm cannon is angled downwards, as seen in Figure 8, by default. This is because the orientation of the controllers is at an angle, and to line

up the gun with the players arm, it needs to be offset relative to the controller position.

The player has a variable hitbox, so that they can crouch down behind cover and not be hit by enemy's shots.

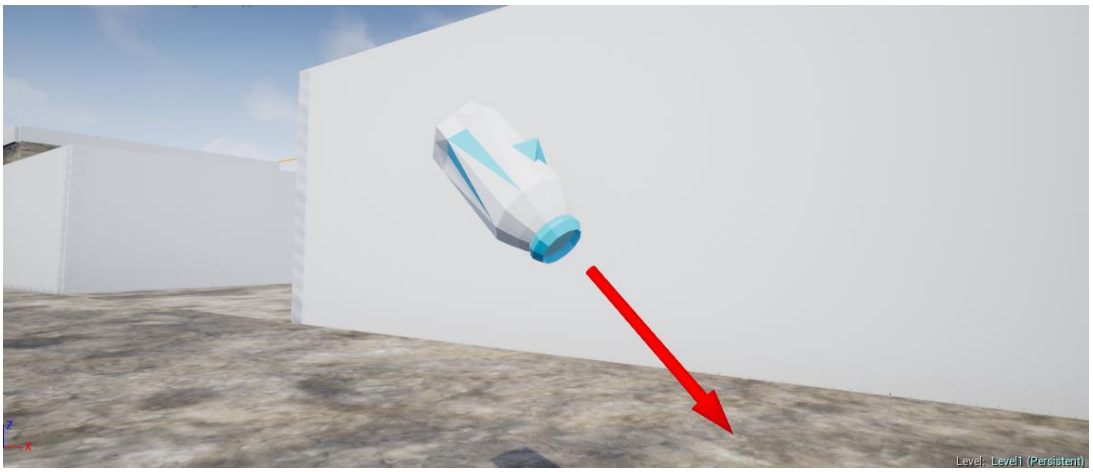


Figure 8: A picture of the arm cannon used in game.

4.4. Level Design

The level design is linear and designed to have the player move about. As mentioned in Chapter 3, the level design would be focused around making the users move about in order to flex the movement methods. The final level can be seen below.

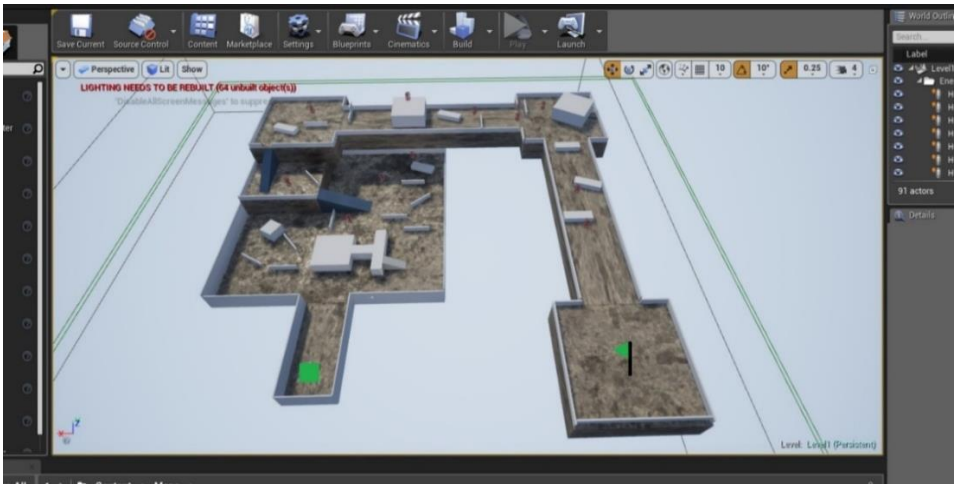


Figure 9: An overhead view of the final level created.

The user would be started on the left-hand side, indicated by the green square, and would have to reach the goal indicated by the flag.

The main reasoning for this implementation is to expose the movement methods to different scenarios and contexts. Verticality is included to test how the movement methods feel when moving on the Y plane, since most games of the future are unlikely to stay on same level the entire game. The end of the level is

more long distance, to see how the movement fares when moving for a long period of time, while the beginning is more akin to a maze to see how the movement feels when constantly changing direction.

4.5. Tools

4.5.1. Game Engine – Unreal Engine 4

The game was implemented in Unreal Engine 4 (UE4) by Epic Games. The language of Unreal Engine 4 is C++, which excels in performance and speed. UE4 is my preferred tool and the engine I am most proficient in.

It is a "complete suite of creation tools" for creating video games at an industry standard [25]. Natively having support for the development of VR games and the HTC Vive makes it an ideal choice for this project.

Another more common option for games development is Unity by Unity Technologies. It uses C# instead of C++, which is substantially friendlier to work with from a programming perspective. While considered easier to develop with, I opted not to use it. This is because of my preference to UE4's traditional Object-Oriented design, where all of the game objects exist as classes, as opposed to Unity's "prefab" design, where game object exists as "prefabs" having scripts attached to them. Using UE4 meant I wouldn't have to learn a whole new methodology of programming and would save time on development.

Regardless, UE4 utilizes classes as objects, where an object will have instance variables, methods and properties associated with them.

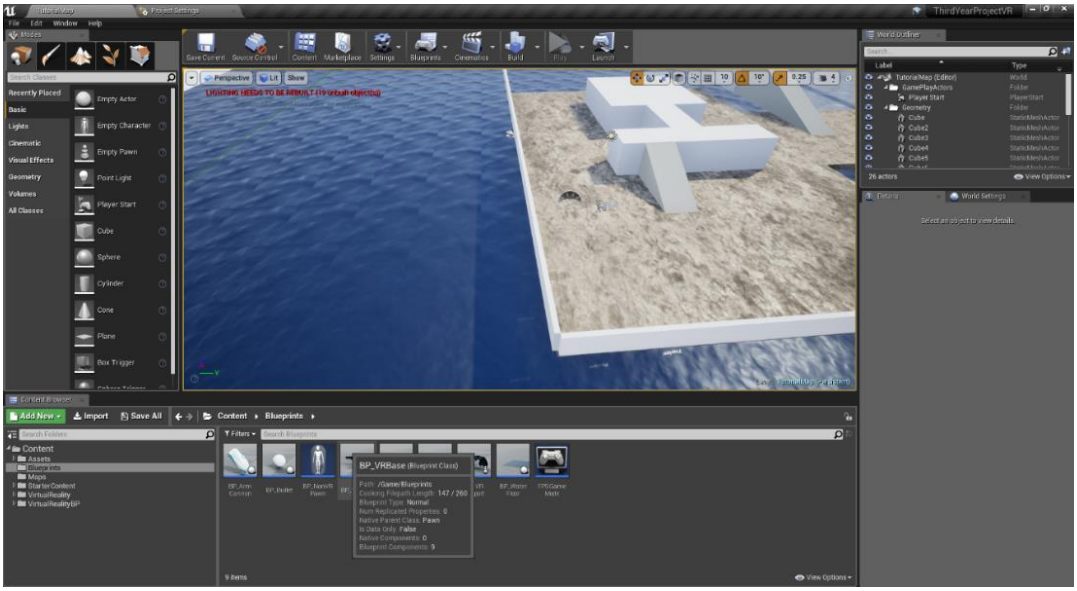


Figure 10: Screenshot of Unreal Engine 4 interface

UE4 automatically sets up components such as scene and level structure, allowing me to create levels with the building blocks they provide. Objects and

file structure can be easily handled, as seen in Figure 10. Objects created in UE4 can then be placed in these levels alongside the geometry. A hierarchy of all the objects in the level can be seen on the right, for easy management.

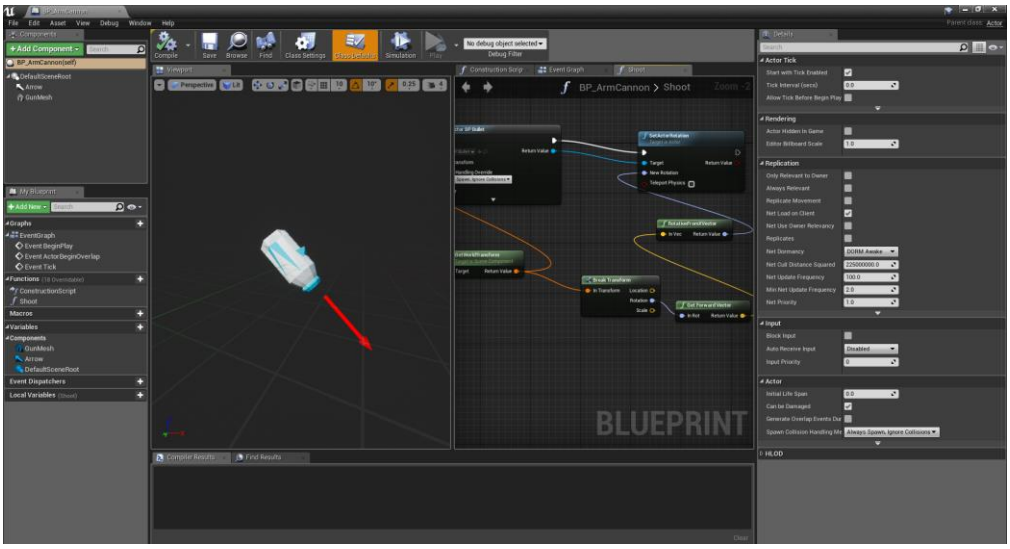


Figure 11: Basic class design interface in UE4, current gun model shown

Noticeably from Figure 11, code can be drawn and visualised with the UE4 Blueprints scheme. Though the code is in C++, the visualisation makes it much easier to understand and absorb.

4.5.2. Head Mounted Display – HTC Vive

The choice of head mounted display for this project is the HTC Vive. It's the optimum choice due to it having native support for room scale tracking reality and motion controllers. In addition to this, it is supported by my engine of choice, making it easy to develop with.

Other options included the Oculus Rift, Samsung Gear VR and Google Daydream for Android. Gear VR and Daydream were sub-optimal for this project, as they were designed for the Android platform and lacked any input or room-scale movement. Oculus Rift would have been a usable tool for development, but as I already own a Vive, there was no point in trying to obtain a Rift when they are capable of the same feats functionally.

However, it can be assumed that because my implementation of a VR solution only extends to PC use, my solutions may be invalid on Android platforms and headsets which are less capable than the Vive.

4.6. Problems Encountered

During this project, there were problems as developing for VR proved a difficult task, as I was using a medium that isn't commonly developed for.

UE4 has sparse documentation compared to other engines such as Unity. Many seemingly important functions were only briefly described when they needed more clarification. An example of this would be the head mounted display "Get

Orientation and Position" [26], which returns a position and rotation in the form of a Vector and a Rotator, however it is not noted where these returned values are relative to. I ended up using trial and error to discover that the position returned is local relative to the centre of the room on the floor, and is local to the object itself, rather than to the world.

Much of the documentation was like this, with the requirement of carrying out small tests to discover what the functions did, which made development much slower than it needed to be.

During the creation of the Vehicle-Based movement method (Section 4.1.3), I encountered problems with creating a solution. Firstly, having to create a frame of reference in the form of a mechanized suit that the player was in was difficult. No models pre-existed for me to use, so I had to create one myself using a 3D modelling program, which isn't ideal. After simulating the concept of the player existing inside the vehicle, I had difficulty orienting the vehicle properly in gameplay. Initially, I planned to have the vehicle automatically orient on the players movement, but because most VR head mounted displays only have three tracking points available this was not viable. As a result, the controls for the vehicle are manual.

5. Testing

Tests were designed to ensure that the demonstration was working properly. This was mainly to prepare it for the user evaluation. Testing in games development is difficult, as bugs are much harder to locate, despite being easy to reproduce.

5.1. White Box Testing

White Box testing is testing an application and developing knowing of the internal structures, as opposed to Black Box which works on functionality.

By nature of the application, unit tests are not applicable to gaming applications. Unit tests usually imply that a given input to a program or application will intend to return a certain value.

The methodology to white box testing included extensive Iterative Testing as development progressed, mentioned in Section 7.2. This is described as a development style where once a feature is implemented, it is tested and then improved, simply on practice [27]. Using this methodology, I was able to improve my solutions by testing them myself. This can be seen, for example, where initially I created a smooth movement type from Section 2.4.1 and then extended it to work for the Vehicle-based movement type by adding the "robot/shell". I ensured the movement method was first customizable and usable by itself, in programmatic terms, before augmenting it.

Debugging was constantly used in development to give more in-depth feedback about what was going on in the program. Since a lot of the control flow in a game powered by Unreal Engine is obscured, print lines were used frequently for cases such as determining what classes were executing what code at what time. Games programming uses vector mathematics heavily for recording the positions, rotations and scales of objects, especially with VR. Being able to see what values the HTC Vive's controllers were feeding back the software was significantly useful in debugging.

Tools such as line traces were extremely useful in debugging functionality. In UE4, line traces can be inserted and used anywhere, similarly to a console log in a typical programming language. They are drawn visually in game.

As seen in Figure 12, these line traces were useful for marking positions and movements in VR space.

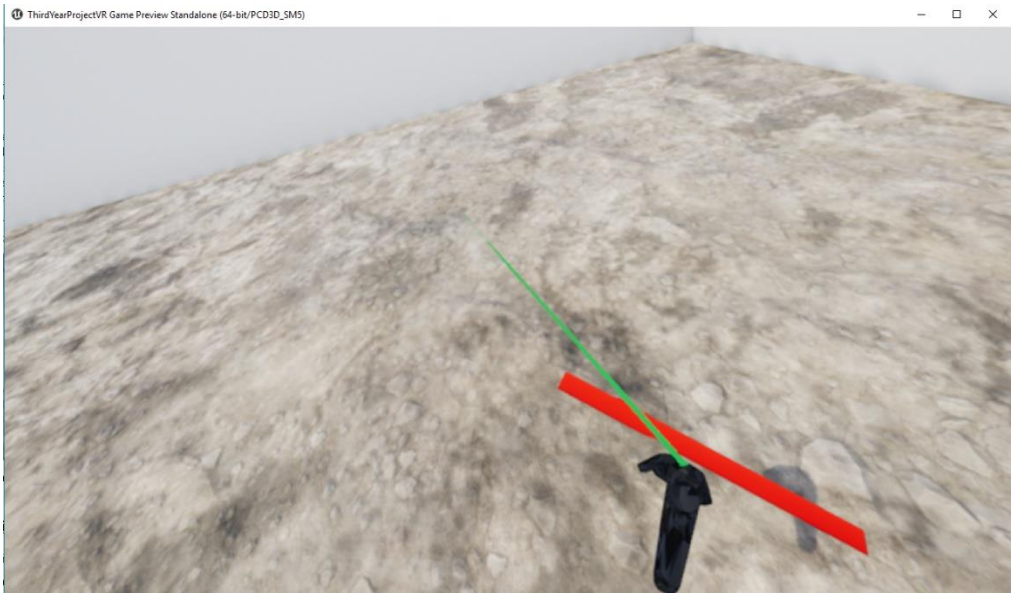


Figure 12: A green debug line in game, being drawn from the Kinetic Movement method, illustrating the controllers previous position when the button is pushed to where the controller currently is.

5.2. Black Box Testing

Black box unit testing is the concept of testing an application from the outside. The tester isn't aware of the inner workings of the application, only their input and output. This is the type of testing most applicable to video games, as they are heavily interactive compared to normal software. Validation on the testing is mainly done on the functional requirements of the application [28].

Test Number	**Summary**	**Steps**	**Expected Result**	**Actual Result**
1	Can the player move with the teleportation method	Use the teleport function by holding the face button, pointing to destination and releasing	Player teleports to the intended destination	Player teleports to the intended destination
2	Can the player teleport to unintended locations	Use the teleport function to try teleport into walls and ceilings	Player cannot teleport into these places	Player cannot teleport into these places
3	Can the player use the kinetic movement	Hold Face button down and move hand, the player should be moving themselves relative to the hand movement	Player moves correctly and with the movement of the hand	Player moves correctly and with the movement of the hand
4	Can the player move using the	Use thumb on the Face button, the	Player moves smoothly	Player moves smoothly

	Vehicle-based movement	player should move in the direction relative to position of the thumb from the centre of the face		
5	Using kinetic and Vehicle-based movement, can the player traverse up slopes and stairs	Same as case 4, but using the movement on slopes and stairs	Player can move up slopes and stairs	Player can move up slopes and stairs
6	Do all movement methods work on the HTC Vive	Run the game using the HTC Vive, within editor in Unreal 4.	The game opens and function as stated in tests 1-5.	The game should open and function as stated in tests 1-5.
7	Enemy will chase and shoot player once they enter a certain radius	Player should move towards enemy, in front of enemy's view	Enemy should turn to shoot at the player and move towards them	Enemy does turn to shoot at the player and move towards them
8	Game achieves an appropriate average framerate	Run game in editor with Vive attached and measure the framerate over a minute	Average framerate should be around 90fps	Average framerate is around 115fps

Table 3: Black Box functional tests table

These tests can be used to confirm that the functional requirements defined in Section 3.2. Test numbers 1-5 confirm that FR3 and FR4 are implemented fully and Test 6 conforms with FR1 and FR2.

Test 8 is a benchmark that conforms with NFR1, seen in Figure 13, and in large in Appendix E. Using Unreal Engine's "stat gpu" command, the average and maximum amount of time a frame is rendered in can be viewed, along with the different components of the game that are causing the load times. As shown in the image below, the average time for a frame to be rendered is 8.64ms, which averages 115fps. This means the technical demonstration is far above the required average framerate for VR games.

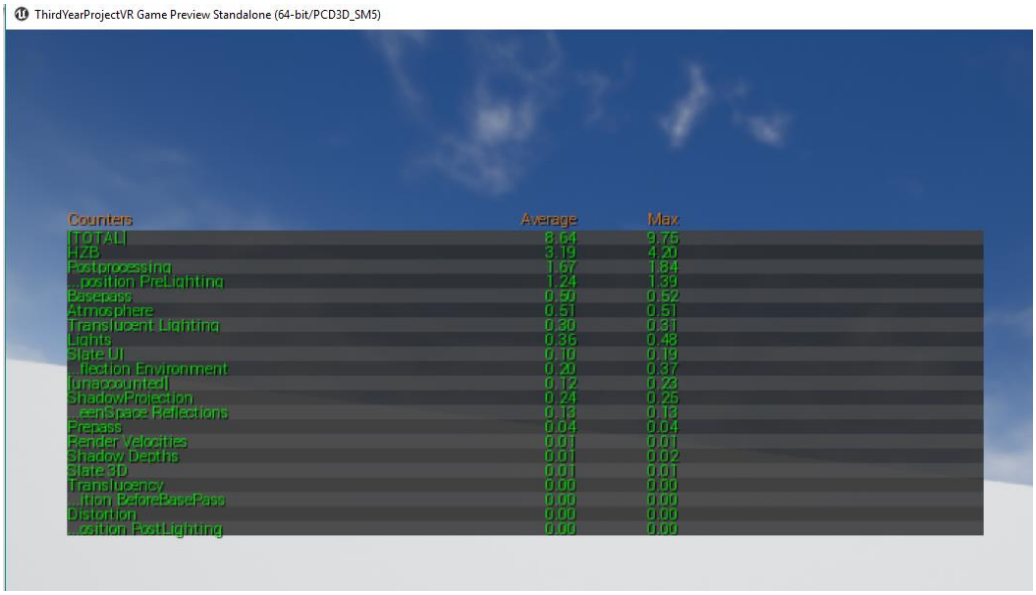


Figure 13: The test GUI showing the time in milliseconds for a frame to be rendered.

6. User Evaluation

User testing was the best way to evaluate whether the solutions I created were valid, as ultimately a solution is only valid if approved by users. By conducting an experiment where users use the solutions I have created, I can evaluate whether they're effective, using the user feedback.

6.1. User Evaluation Methodology

For the user testing, I had 10 people play my demonstration, with all different movement types. The ERGO submission number was 40298.

After briefing the participants on what they would be doing in the study, the participants would play the level three times over with the movement types. The movement methods would be played in a random order, as it can be assumed that playing the same level multiple times would have participants get better at the game as they learn how it works. This has the possibility of skewing data, where it may seem like one movement method is vastly outperforming the others.

Each time they played through the level, I made notes and recorded any significant quotes about the movement methods or testing experience, along with timing of how long it took them to complete a level with a certain movement method.

Afterwards, users would fill out a questionnaire with the following questions:

- Rank the movement methods from 1 – 3, where 1 is the type you liked the most, and 3 is the one you liked the least
- Why did you rank the movement methods as you did?
- Were any of the movement methods difficult to use, in context of playing the game? If so, why?
- Rank the movement methods from 1 – 3, where 1 is the type that felt the most comfortable using, and 3 the least comfortable using
- Why did you rank the movements as you did?
- What changes would you make to any of the movements to suit them better to yourself?
- Final comments, anything you'd like to say about the experience?

This allowed me to collect a significant amount of data from each user, including qualitative and quantitative data. Mainly, it was important that data on what users wanted out of the movement methods was collected, and whether the solutions created satisfied those wants.

Using this data, I can assimilate the best-favoured movement method in terms of ease of use and comfort for motion sickness. The qualitative data can be used to home in on specific problems faced in the VR environment and perhaps narrow down a generic solution.

6.2. Results

These are the results of the user evaluation. I have compiled graphs out of the way the participants ranked the different movement methods and included different quotes from participants. The full comprehensive list of results from the survey given to the participants can be found in Appendix D.

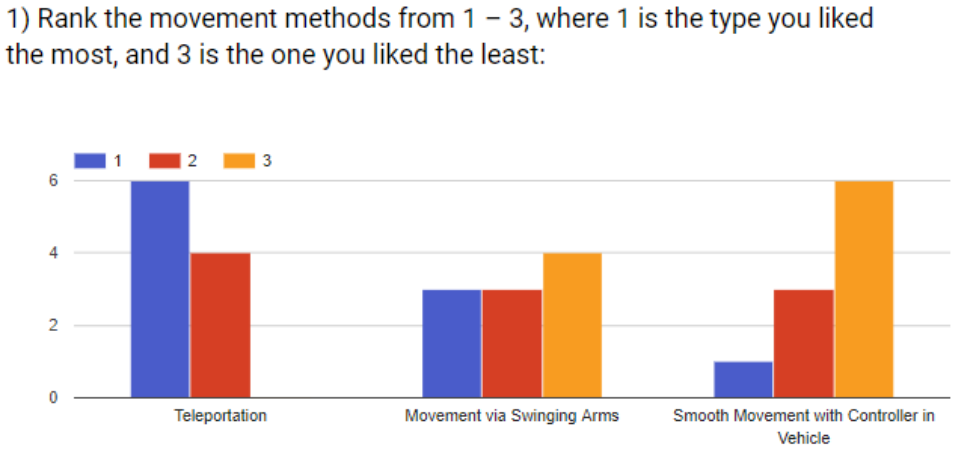


Figure 14: Graph showing results of participants favourite movement methods, ranked

Firstly, the most interesting thing about the rankings in Figure 14 is that no one ranked Teleportation as the one they enjoyed the least, and only one participant ranked Vehicle-based movement as their favourite. The kinetic based movement seemed to be in the middle of the two, with roughly equal rankings by all participants.

When the users were asked why they ranked the movement methods as they did, participants gave a variety of answers. Participant 1 said “The vehicle was very frustrating. It blocked my field of vision and if I moved in the real world, the vehicle would block my vision in the virtual world.” 5 and 9 also claimed that the vehicle was actually very frustrating to use from a gameplay point of view.

It was generally agreed that teleportation was the easiest and fastest to use. “Speed and ease of use of teleporting was very good to use” was claimed by Participant 7.

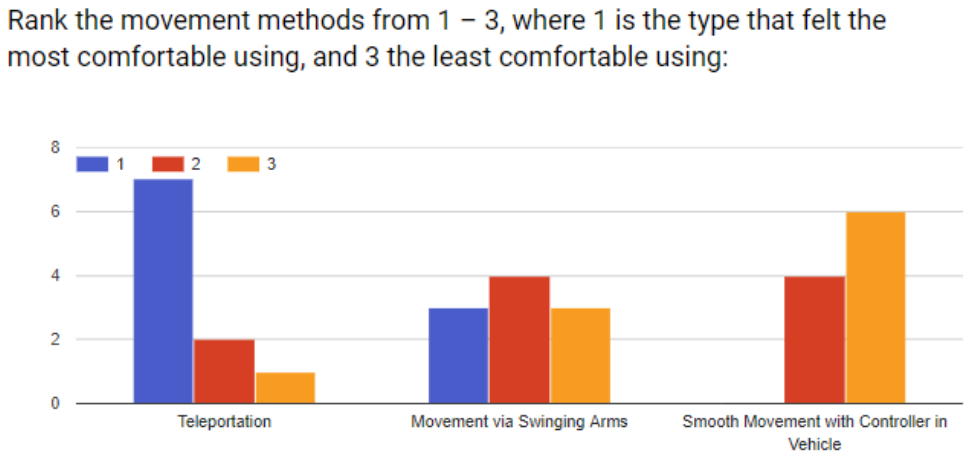


Figure 15: Graph showing what methods were ranked as most comfortable, in terms of motion sickness

Figure 15 shows that none of the users found the vehicle-based movement better than the other two solutions. However, Participants after trying the Smooth movement without the vehicle said they realised that the vehicle was, in fact, useful. Participant 10 said “Smooth movement screen drag wasn't immersive, without vehicle made me feel wobbly” after trying out smooth movement without the vehicle.

Otherwise, it’s mostly agreed that the teleportation movement is the most comfortable to use, with the least motion sickness. It isn’t without it’s problems, since Participant 1 “felt like I did not have full control or autonomy over my movements”. Others said that teleporting required them to reorient themselves with the game world afterwards.

When asked how they would personally improve or change any of the movement methods, most of the replies were disjointed with no correlation. The one agreed opinion was from 3, 7, 8 and 9 wanting the Kinetic movement to work faster, with a higher ratio of arm movement to game movement. Participant 2 would have liked for smooth movement to be merged with teleportation and a few of the participants would have liked the Vehicle part of Vehicle-based Movement removed.

Regarding the timings of the participants, it seemed that in all situations the participants would improve level completion time every time they tried a new movement method, making the data not very useful.

6.3. Analysis and Evaluation

Based on the results from above, there are some clear correlations and conclusions that can be drawn.

Teleportation was the most favoured technique, because of it’s ease of use and lack of motion sickness associated with it, for the most part. Some participants said that it would take time to get used to, since there is less precision of

movement compared to the other two methods. Others had the opinion that while it was much easier to use than the other methods, it was largely inappropriate for the game they were given. They could not imagine playing a large game using it, for a long period of time, especially a multiplayer one. In addition to this, many participants disliked the fact that they'd have to reorient themselves after teleporting.

The vehicle-based testing was less favoured as the vehicle was inhibiting rather than useful. However, when tested against typical smooth movement as described in Section 2.4.1, it's reception was much better at reducing motion sickness. Participants described it as annoying and tedious, since control of the vehicles orientation had to be taken into account while trying to move around and complete the level. Arguably, as the weakest movement method, my implementation of the vehicle may be the reason for all of this.

Kinetic based movement generally received complaints that, while no sickness was occurring from it, the movement was tedious to perform continuously. Participants felt that the movement was, while precise, much slower than the other two movement methods for the effort required to make the movement happen. Otherwise, there seems to be no indication that the kinetic movement caused significant motion sickness.

The results mainly prove why the teleportation method has been the defacto default movement method in VR gaming. In context of suitability for a game, even participants understood it wasn't built for a shooter like game but enjoyed using it because it was easy to use. However, the other two methods still showed promise to some degree.

For the kinetic movement, it seemed that having a higher sensitivity and movement speed was more favourable. Since the only complaint about it was that it was tedious to use, perhaps simply reducing the amount of effort to move with it would make it a suitable movement method to replace teleportation. More user evaluation would be required to see whether an increase in sensitivity would reintroduce motion sickness to the movement method, however.

The vehicle-based movement can work if the implementation is improved. Since it did seem to improve the motion sickness of the participants compared to regular smooth motion, perhaps the improvement of the vehicle control would make it more useable. This would be made easier if the vehicle automatically oriented itself around the user, rather than the user doing it manually. This would require more precision from the hardware, since the Vive only tracks the head, left hand and the right hand. If there were trackers on the shoulders or waist, then orientation of the body could be tracked independently of the head.

Overall, it can be concluded that while the other two proposed movement methods could not dethrone teleportation in this study, they may be able to do so with some iteration. Conversely, it's possible the "best" movement method may be unattainable largely since susceptibility to motion sickness and how people prefer their gameplay is a case by case problem.

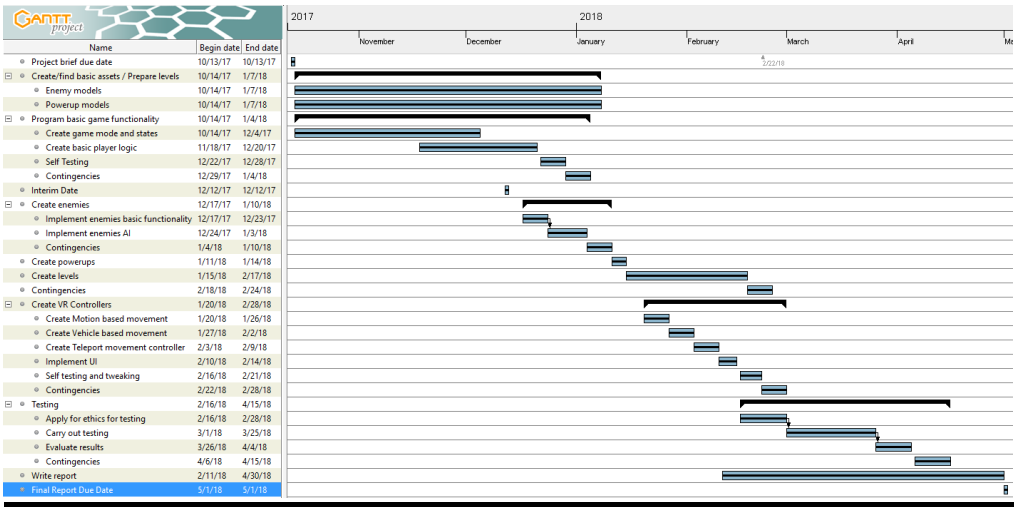
7. Project Management

This section will cover how the project was managed in terms of planning and risk analysis, along with programming methodology. It will also evaluate how the project ended up executing in comparison to the original plan.

7.1. Project Gantt Chart

The project roadmap is outlined in a Gantt Chart within Figure 16 and Appendix C. It shows my plan of development through the project, including contingencies. It shows the completion times and lengths of each task as they happened.

The Gantt chart was to serve merely as a guideline for overall progress, as Iterative Programming was used in software development during its main sprints. It was useful for ensuring the project stayed roughly on track and provided a timeline for major developmental milestones.



Name	Begin date	End date
Project brief due date	10/13/17	10/13/17
Create/find basic assets / Prepare levels	10/14/17	1/7/18
Enemy models	10/14/17	1/7/18
Powerup models	10/14/17	1/7/18
Program basic game functionality	10/14/17	1/4/18
Create game mode and states	10/14/17	12/4/17
Create basic player logic	11/18/17	12/20/17
Self Testing	12/22/17	12/28/17
Contingencies	12/29/17	1/4/18
Interim Date	12/12/17	12/12/17
Create enemies	12/17/17	1/10/18
Implement enemies basic functionality	12/17/17	12/23/17
Implement enemies AI	12/24/17	1/3/18
Contingencies	1/4/18	1/10/18
Create powerups	1/11/18	1/14/18
Create levels	1/15/18	2/17/18
Contingencies	2/18/18	2/24/18
Create VR Controllers	1/20/18	2/28/18
Create Motion based movement	1/20/18	1/26/18
Create Vehicle based movement	1/27/18	2/2/18
Create Teleport movement controller	2/3/18	2/9/18
Implement UI	2/10/18	2/14/18
Self testing and tweaking	2/16/18	2/21/18
Contingencies	2/22/18	2/28/18
Testing	2/16/18	4/15/18
Apply for ethics for testing	2/16/18	2/28/18
Carry out testing	3/1/18	3/25/18
Evaluate results	3/26/18	4/4/18
Contingencies	4/6/18	4/15/18
Write report	2/11/18	4/30/18
Final Report Due Date	5/1/18	5/1/18

Figure 16: Original Gantt Chart

Unfortunately, this Gantt chart did not pan out as it was meant to as the project went on towards the end. Changes, that reflect how the schedule ended up executed, to the Gantt chart are shown in Figure 17.

One cause of change was the difficulty encountered with creating the vehicle-based movement method, which ended up taking much more time to create than anticipated.

The biggest change to derail the project was the time it ended up taking to have my user evaluation validated by ERGO. Despite submitting for ethical approval on time, with expected wait times of up to two weeks, ERGO took over 2 months to complete my request. Furthermore, I had to resubmit my case amended to their standards afterwards, since the original submission was not approved. This caused my user evaluation phase to be shortened, where instead of taking place from mid-March to the end of April, the evaluation was only approved to happen from mid-end of April.

In retrospective, the ethical approval should have started earlier, and it was advised that I started it during Semester 1 rather than Semester 2. Since the

approval task requires me to wait for someone else and usually happens multiple times, this outcome should have been considered. Additionally, the three movement methods would have been better developed if created at the beginning of the project rather the end, since it would have been easier to tune them over a longer period of time.

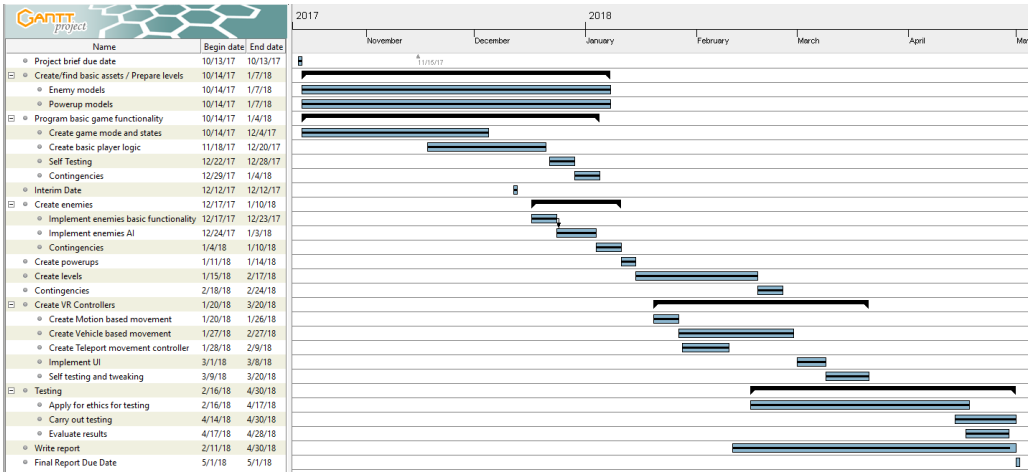


Figure 17: Real Gantt Chart

7.2. Iterative Programming

Iterative Programming was used as the development cycle in this project. It took place during the phases of developing the core game and its mechanics, along with the movement systems themselves. It allowed me to break down the development of a large application like this into smaller chunks, where I could implement one feature at a time and test the functionality of it regularly [29].

The use of Iterative Programming can be seen in the original Gantt chart, where different tasks will be implemented one after another. However, the nature of game development is that many of the mechanics and features are dependent on each other for testing and implementation. I could not implement the shooting feature of a gun if I do not have a character developed to shoot the gun. This meant that some tasks were worked concurrently for the project to be developed evenly.

This can be seen, for example, in my Gantt chart where creation of levels began before the completion of the base game. This was because the base game could not be tested without levels.

Overall, through the project, development itself went smoothly. Aside from the vehicle-based movement, all features developed went as according to the Gantt chart and plans.

7.3. Risk Analysis

Although risks cannot be entirely avoided, they can be managed and minimized. Identifying potential risks is paramount for ensuring the project runs as smoothly as possible. [30]

Type	**Description**	**Probability (High, Medium, Low)**	**Impact (H, M, L)**	**Avoidance Strategy**
Human Risk	Illness	M	M – Performance reduced when ill	Contingency time has been allocated for tasks
Technical Risk	Work lost	L – Hardware breaks, or accidentally break project during development	H – Tasks will have to be redone if not backed up properly	Work shall be backed up on GitHub and DropBox, along with both being on laptop and desktop
Technical Risk	Hardware breaks	L – Computer or HTC Vive could break.	H – Development will cease if I don't have the hardware to develop with	I have a backup for computer components, and the University will let me borrow a Vive if needed
Project Risk	Schedule unlikely to be followed entirely	H – It's unlikely that the schedule can be followed 100%	M – In extreme conditions, the schedule can be entirely derailed, making the planning useless	Adjust the schedule and tasks during the project to ensure that they all have reasonable times for completion
Project Risk	Tasks taking too long	M – Some tasks may take longer than planned, and overrun the schedule roughly	H – Derailing the plan may cause other tasks to be rushed to completion	Contingency time has been allocated for tasks
Project Risk	ERGO study not being approved	H- It is common for ERGO studies to take up to 2 weeks to submit with multiple submissions before approval	M – If it takes too long to submit	The submission will have to be submitted early. Over a months' notice given, as seen in the original Gantt chart.

Table 4: Risk Analysis Table

For each risk, I had created an avoidance strategy to solve its potential problem, should they occur.

Most of my solutions involved the use of contingency time to consider that accomplishing tasks may take longer than thought. Using Git and GitHub, I could

keep track of previous versions and ensure I had a backup of the project if anything went wrong during development.

In reality, only one risk ended up occurring during the project which was the ERGO study approval, from above. Unfortunately, this occurrence was an edge case where the approval took over two months to complete, and mitigation was difficult. Luckily, I had enough time to execute user evaluation due to contingency times, but in the future this event will be noted for risk analysis.

8. Conclusion and Future Work

This third-year project turned out to be more difficult to implement than expected and creating the implementations and running user evaluation was challenging. For future projects, the problems mentioned in Chapter 7 will be taken into consideration when planning, as contingencies were not thought of enough.

I set out to find a solution to the movement problem in the Virtual Reality climate, which involved dethroning teleportation movement as the default movement technique in games. Though I did not find a conclusive solution, I believe the research conducted will be useful in creating a solution in the future.

As mentioned in the user evaluation, the solutions were satisfactory in solving VRs most paramount problem, motion sickness. Most users experienced minimal motion sickness with the newly implemented methods compared to conventional methods such as smooth motion. However, these new methods weren't polished enough to be useable fully within a game.

For future work within the scope of this project, iterating over the current solutions and repeating user evaluation with a larger sample size could eventually yield a conclusive solution. Though it would only be a solution for the given type of game it's tested for, this game type is highly mobile and hopefully applies to other games.

It is hoped that the research that has taken place, in the creation of this report, is useful in the future of Virtual Reality and its usage in the games industry in the future. Perhaps these methods can be extended to application usage rather than just games too. Mainly, the reduction in motion sickness discovered in my solutions could help the VR development industry in general.

References

[1] S. Steinberg, “VIDEOGAME MARKETING AND PR Vol. 1: Playing to Win,” 2007.

[2] T. Fullerton, *Games Design Workshop*. 2008.

[3] J. Schell, *The Art of Game Design*. 2008.

[4] R. Pausch, R. Gold, T. Skelly, D. Thiel, and T. Hall, “What HCI Designers Can Learn From Video Game Designers,” 1994.

[5] J. Steuer, “Defining Virtual Reality: Dimensions Determining Telepresence,” *J. Commun.*, vol. 42, no. 4, pp. 73–93, 1992.

[6] Google, “Get Cardboard – Google VR.” [Online]. Available: https://vr.google.com/cardboard/get-cardboard/. [Accessed: 21-Apr-2018].

[7] “Oculus Rift | Oculus.” [Online]. Available: https://www.oculus.com/rift/#oui-csl-rift-games=robo-recall. [Accessed: 21-Apr-2018].

[8] “VIVE™ United Kingdom | Buy VIVE Hardware.” [Online]. Available: https://www.vive.com/uk/product/. [Accessed: 21-Apr-2018].

[9] Valve, “GitHub - OpenVR,” *GitHub*, 2018. [Online]. Available: https://github.com/ValveSoftware/openvr.

[10] J. J. LaViola Jr, “A discussion of cybersickness in virtual environments,” *ACM SIGCHI Bull.*, vol. 32, no. 1, pp. 47–56, 2000.

[11] A. Rolnick and R. E. Lubow, “Why is the driver rarely motion sick? The role of controllability in motion sickness,” *Ergonomics*, vol. 34, no. 7, pp. 867–879, 1991.

[12] J.-W. Lin, H. B.-L. Duh, D. E. Parker, H. Abi-Rached, and T. A. Furness, “Effects of field of view on presence, enjoyment, memory, and simulator sickness in a virtual environment,” in *Virtual Reality, 2002. Proceedings. IEEE*, 2002, pp. 164–171.

[13] J. M. Oscillada, “Comparison Chart of FOV (Field of View) of VR Headsets – Virtual Reality Times,” *2017*. [Online]. Available: http://virtualrealitytimes.com/2017/03/06/chart-fov-field-of-view-vr-headsets/. [Accessed: 18-Apr-2018].

[14] J. D. Prothero, M. H. Draper, T. A. Furness 3rd, D. E. Parker, and M. J. Wells, “The use of an independent visual background to reduce simulator side-effects.,” *Aviat. Space. Environ. Med.*, vol. 70, no. 3 Pt 1, pp. 277–283, 1999.

[15] “Teleportation Sucks...and we can do better. — VRemedy Labs.” [Online]. Available: https://vremedylabs.com/blog/2017/7/30/teleportation-sucksand-we-can-do-better. [Accessed: 12-Dec-2017].

[16] “Loco-motion: The ridiculous state of VR movement — VRemedy Labs.” [Online]. Available: https://vremedylabs.com/blog/2017/7/31/loco-motion-

the-ridiculous-state-of-vr-movement. [Accessed: 12-Dec-2017].

[17] K. Voll, “GDC Vault - This is Your Brain on VR: A Look at The Psychology of Doing VR Right,” *2016*. [Online]. Available: https://www.gdcvault.com/play/1023643/This-is-Your-Brain-on. [Accessed: 12-Dec-2017].

[18] C. Edwards, “Valve Lines Up Console Partners in Challenge to Microsoft, Sony - Bloomberg,” *4/11/2013*. [Online]. Available: https://www.bloomberg.com/news/articles/2013-11-04/valve-lines-up-console-partners-in-challenge-to-microsoft-sony. [Accessed: 03-Dec-2017].

[19] J. W. Murray, *Building Virtual Reality with Unity and Steam Vr*. CRC Press, 2017.

[20] Unknown, “PSVR Rids VR of Screen Door Effect (SDE) : PS4,” 2016. [Online]. Available: https://www.reddit.com/r/PS4/comments/51vk72/link_psvr_rids_vr_of_screen_door_effect_sde/. [Accessed: 30-Apr-2018].

[21] H. Song, J. Kim, and C.-C. J. Kuo, “Real-time encoding frame rate control for H. 263+ video over the Internet,” *Signal Process. Image Commun.*, vol. 15, no. 1–2, pp. 127–148, 1999.

[22] K. Oxland, *Gameplay and design*. Pearson Education, 2004.

[23] X. Cui and H. Shi, “An overview of pathfinding in navigation mesh,” *Int. J. Comput. Sci. Netw. Secur.*, vol. 12, no. 12, pp. 48–51, 2012.

[24] D. Perez, M. Nicolau, M. O’Neill, and A. Brabazon, “Evolving behaviour trees for the mario ai competition using grammatical evolution,” in *European Conference on the Applications of Evolutionary Computation*, 2011, pp. 123–132.

[25] “Unreal Engine | Game Engine Technology by Unreal.” [Online]. Available: https://www.unrealengine.com/en-US/what-is-unreal-engine-4. [Accessed: 11-Dec-2017].

[26] Epic Games, “Get Orientation and Position | Unreal Engine,” 2018. [Online]. Available: http://api.unrealengine.com/INT/BlueprintAPI/Input/HeadMountedDisplay/GetOrientationandPosition/index.html. [Accessed: 17-Apr-2018].

[27] J. Rubin and D. Chisnell, *Handbook of usability testing: howto plan, design, and conduct effective tests*. John Wiley & Sons, 2008.

[28] S. Nidhra and J. Dondeti, “Black box and white box testing techniques-a literature review,” *Int. J. Embed. Syst. Appl.*, vol. 2, no. 2, pp. 29–50, 2012.

[29] Y. Francino, “What is iterative development?” [Online]. Available: http://searchsoftwarequality.techtarget.com/definition/iterative-development. [Accessed: 09-Dec-2017].

[30] R. N. Charette, *Software engineering risk analysis and management*. Intertext Publications New York, 1989.

Appendix A – Game Design Document

Basic Description

This game is meant to be a basic shooter game at its core.

You are a character, you can move about a map, and you have enemies to shoot. You can die if you get shot too many times. You are in first person, since you are in VR.

Enemy characters will be basic characters, coloured in red. In contrast to the rest of the game, they should be easy to spot. They can shoot at you.

The goal of the game will simply to reach the end of a level, represented by a goal flag/area.

Movement Mechanics

There will be three main movement mechanics:

a) **Teleportation movement**. This is movement typically used in virtual reality games. Pushing the main face pad on the Vive controller will allow the player to teleport to a destination of their choice, up to a certain distance away.
b) **Kinetic movement.** This is a movement technique employed by VR games such as GORN. The movement of the character will be taken from the physical movement of the player. Pushing the face button and making a walking motion with the controller will cause the player to move.
c) **Vehicle-based movement**. The smooth movement is rarely used in VR because its ability to make players motion sick. However, adding the context of a vehicle that the player is in may reduce the motion sickness of smooth movement. Free movement by simply touching on the Vive Face button the direction the player wants to go.

Player Properties

The player will move at an appropriate rate depending on the controller type. This is because forcing the player to slide in at a certain speed with the kinetic movement, for example, may induce motion sickness.

Jumping is still in consideration.

Player will have non-regenerating health and will have to pick up health packs on the ground to replenish it. The players maximum health will be 200, where health packs will replenish 80 health. These numbers have currently been chosen arbitrarily but will be adjusted properly during testing.

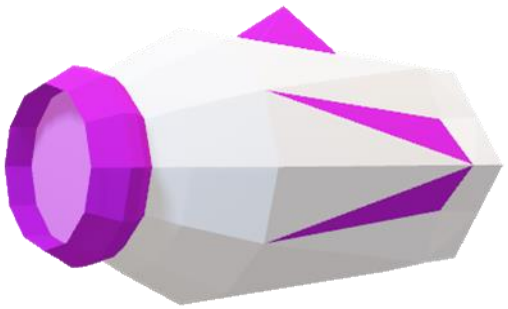
The player's UI / HUD is still in consideration, will have to be tested at implementation. Currently, it's planned to be a health bar at the top left of the players view, which will move with the player, with a simply number floating on top of the gun model, showing the amount of ammo left in the magazine.

Gun mechanics

Player will have an arm cannon for a gun, as it's the easiest to implement to a reasonable standard, because of its flexibility of implied functionality. It's a one-handed weapon that's easy to implement in VR and leaves the hand -primarily used for movement available.

Guns will be projectile based instead of hit-scan based, this is so it will be easier for the player to see where their projectiles are going and dodge enemy projectiles with their movement. Gun damage will be 20, making them able to kill enemies in a reasonable amount of hits.

The player will have infinite ammo, for convenience. Gun model:

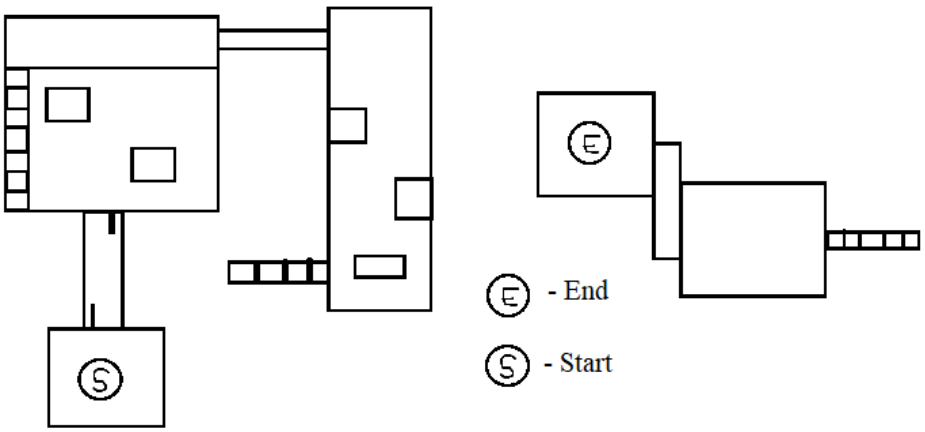


Level Design

The game will use level design that requires the player to move frequently around the map. It will be of linear design. Unlike most VR games, the player will be compelled to move forward and utilize their surroundings.

The player will be confronted with enemies and be required to defeat all the enemies before proceeding forward in some cases. There will be many objects to hide behind to avoid being hit. There will also be powerups placed around for the players to get, which will encourage the use of the movement mechanisms given to them. Here is a sample level blueprint for the first level of the game:

The level will include aspects of verticality, maze like design and long distances. These are to see how usable the movement methods are under ~~most~~ various circumstances.



Enemy Design

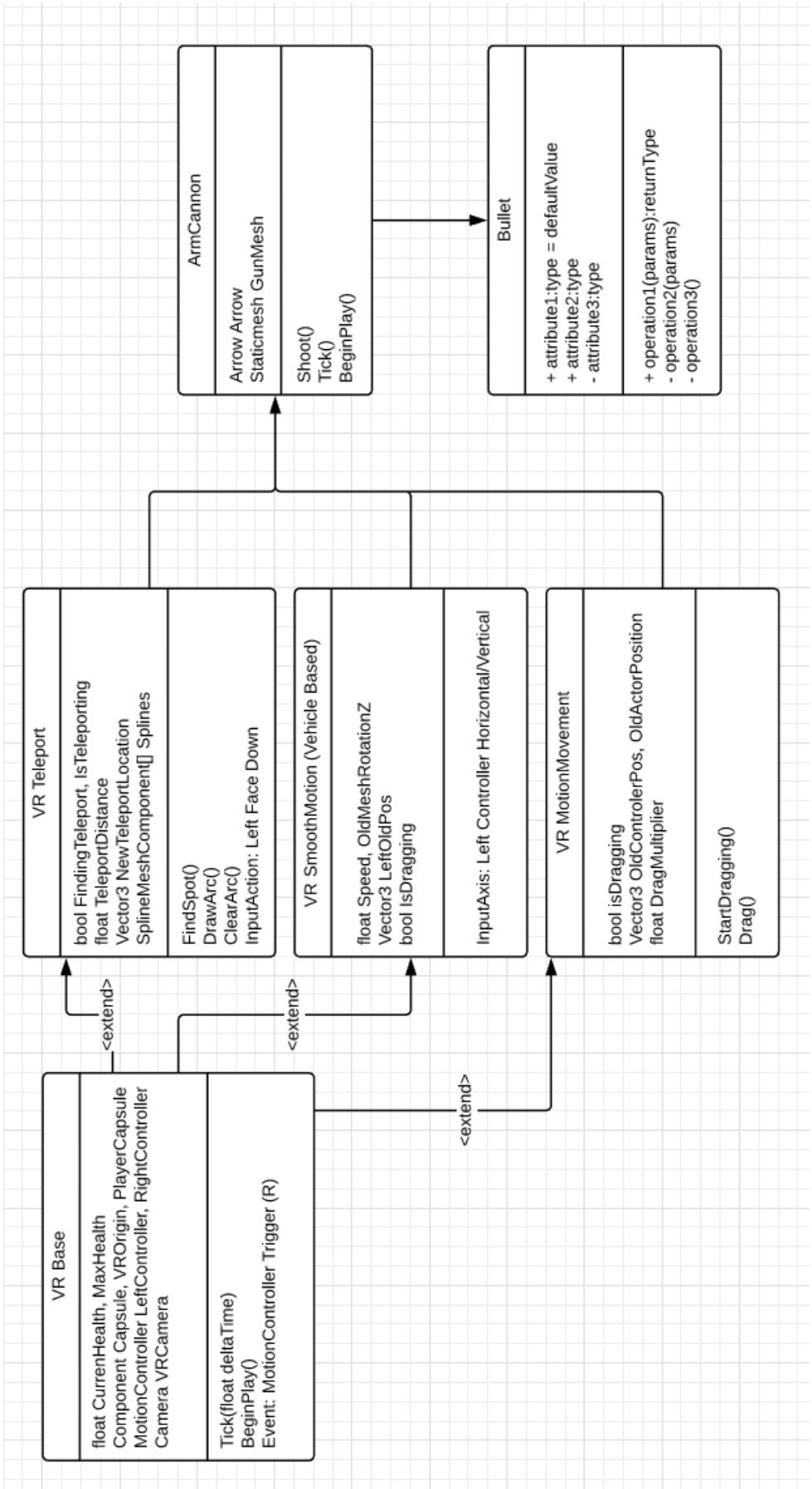
Enemies will be typical humanoids, coloured in red, as said before.

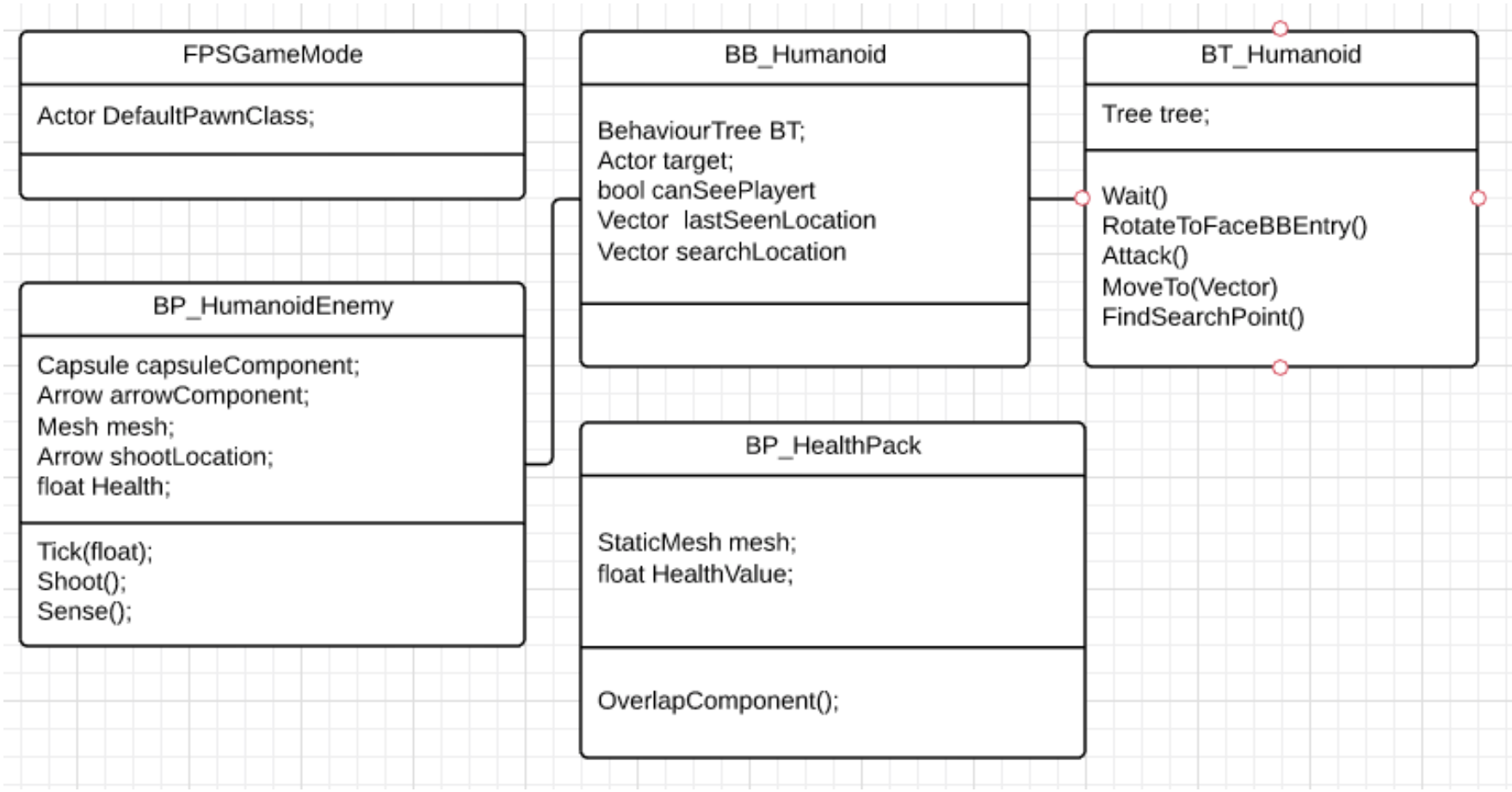
Currently planned, there will be a basic humanoid that can fire the same projectiles as the player, but at a slightly slower speed, so that the player can react to it. Humanoid enemies will have 60 health, making them killable with three shots. However, being hit in the head will induce an instant kill.

There will also be a flying “drone” type enemy that will shoot at the player. This will encourage the player to observe their surroundings more and increases the variety of enemies. Drones will have 40 health, making them killable in two shots.

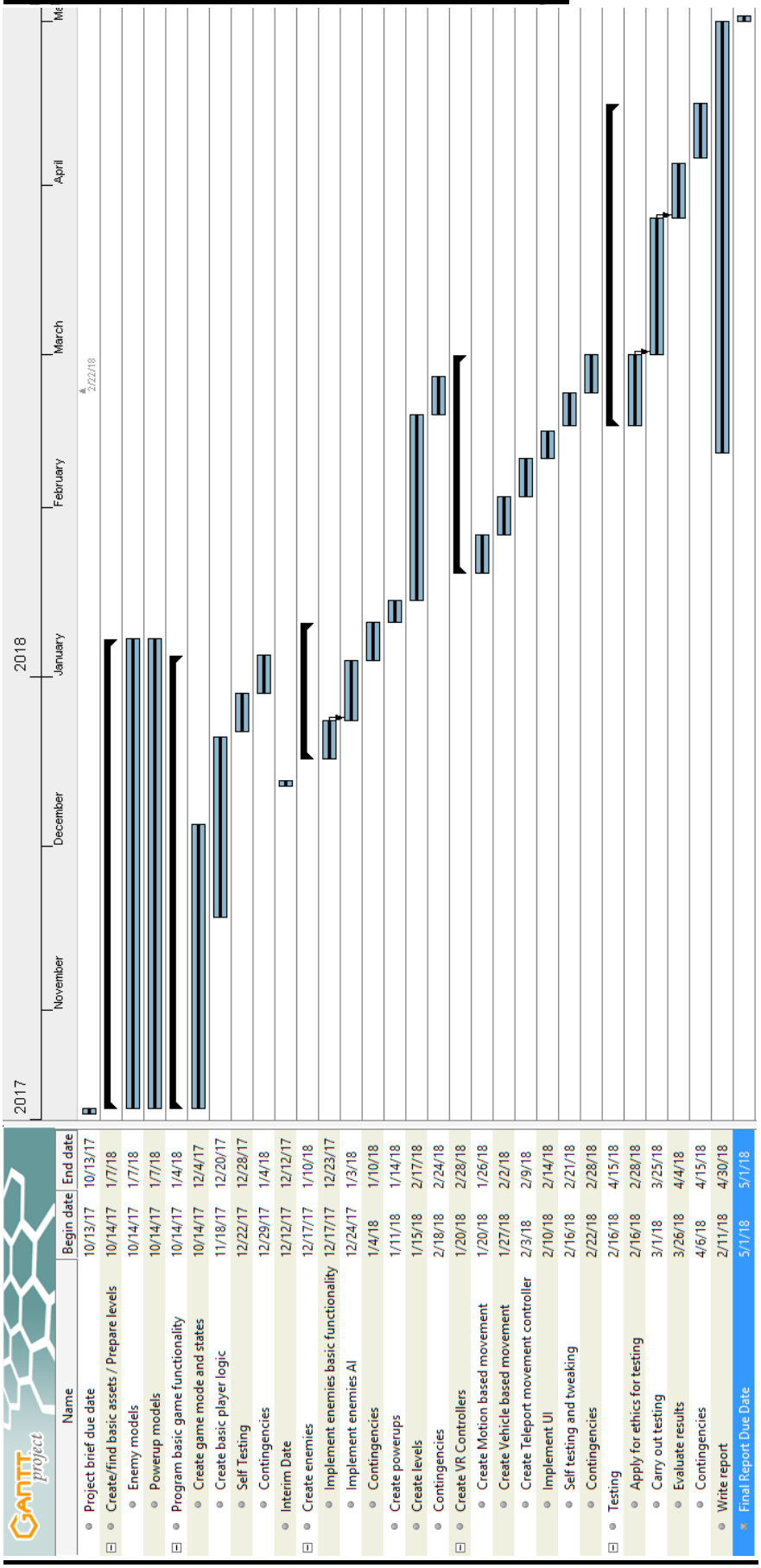
Enemy shots will inflict 20 damages per shot, making them able to kill the player in 10 shots.

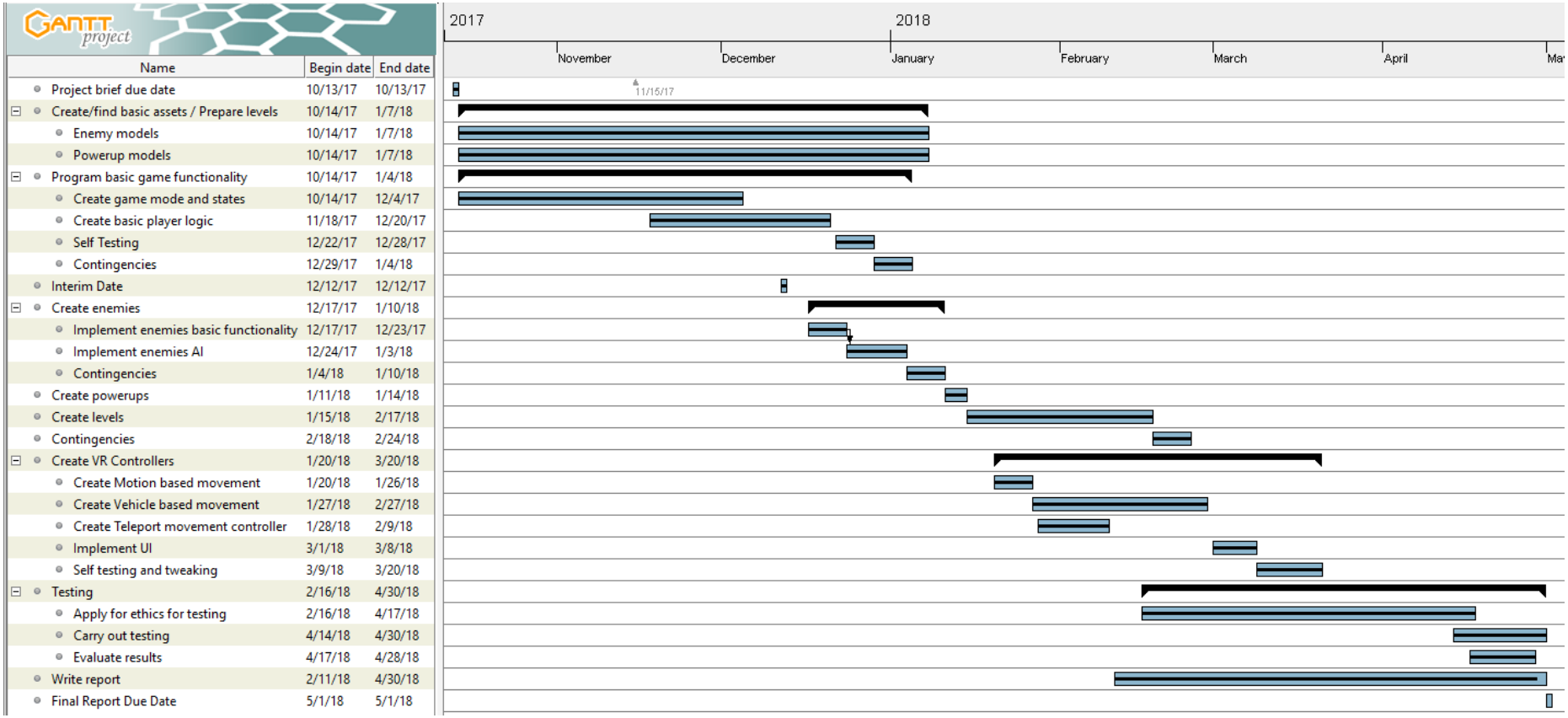
Appendix B – UML Diagrams in large




FPSGameMode
Actor DefaultPawnClass;
BB_Humanoid
BehaviourTree BT;
Actor target;
bool canSeePlayert
Vector lastSeenLocation
Vector searchLocation
BT_Humanoid
Tree tree;
Wait()
RotateToFaceBBEntry()
Attack()
MoveTo(Vector)
FindSearchPoint()
BP_HumanoidEnemy
Capsule capsuleComponent;
Arrow arrowComponent;
Mesh mesh;
Arrow shootLocation;
float Health;
Tick(float);
Shoot();
Sense();
BP_HealthPack
StaticMesh mesh;
float HealthValue;
OverlapComponent();

Appendix C – Gantt Charts in large

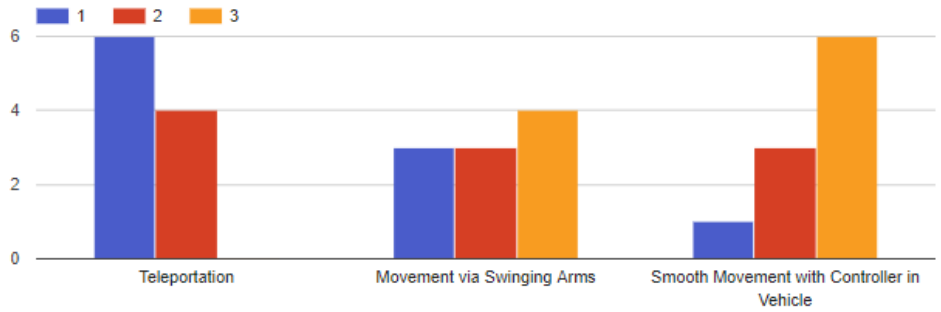




Name	Begin date	End date
Project brief due date	10/13/17	10/13/17
Create/find basic assets / Prepare levels	10/14/17	1/7/18
Enemy models	10/14/17	1/7/18
Powerup models	10/14/17	1/7/18
Program basic game functionality	10/14/17	1/4/18
Create game mode and states	10/14/17	12/4/17
Create basic player logic	11/18/17	12/20/17
Self Testing	12/22/17	12/28/17
Contingencies	12/29/17	1/4/18
Interim Date	12/12/17	12/12/17
Create enemies	12/17/17	1/10/18
Implement enemies basic functionality	12/17/17	12/23/17
Implement enemies AI	12/24/17	1/3/18
Contingencies	1/4/18	1/10/18
Create powerups	1/11/18	1/14/18
Create levels	1/15/18	2/17/18
Contingencies	2/18/18	2/24/18
Create VR Controllers	1/20/18	3/20/18
Create Motion based movement	1/20/18	1/26/18
Create Vehicle based movement	1/27/18	2/27/18
Create Teleport movement controller	1/28/18	2/9/18
Implement UI	3/1/18	3/8/18
Self testing and tweaking	3/9/18	3/20/18
Testing	2/16/18	4/30/18
Apply for ethics for testing	2/16/18	4/17/18
Carry out testing	4/14/18	4/30/18
Evaluate results	4/17/18	4/28/18
Write report	2/11/18	4/30/18
Final Report Due Date	5/1/18	5/1/18

Appendix D – User Evaluation Full Results

1) Rank the movement methods from 1 – 3, where 1 is the type you liked the most, and 3 is the one you liked the least:



Why did you rank the movement methods as you did?

10 responses

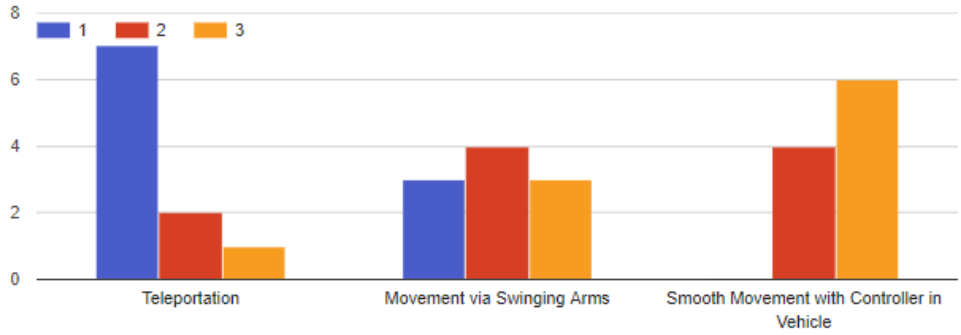
The vehicle was very frustrating. It blocked my field of vision and if I moved in the real world, the vehicle would block my vision in the virtual world. The teleportation felt good but felt like I did not have full control or autonomy over my movements. Movement with swinging arms felt more real life-like, i.e i could pace it how i wanted to, go in all directions, but swinging my arms far to move fast was a bit tiring.
Teleportation allowed for all distances of movement (long and short) which was nice. Loved the smooth movement of the controller in vehicle, but the rotation was a bit annoying and the vehicle restricted your view. Movement from swinging arms was slow, tiring and annoying, but i liked the way it made you travel the entire map.
I rated them based on ease of gameplay
number 1 because it felt like i was more active and that i was actually in the game, it felt like the most realistic. number 2 because it was fun and cover was used a lot, but it was too easy to escape enemies when in trouble. number 3 because it seemed too easy and it seemed like armor. the health bar in number 3 was at the top and i did not like that
The teleportation one was my favourite because it allowed me to move quickly. The smooth movement one was difficult to manoeuvre and made me feel nauseous.
teleportation made it easier to move the user around whereas the movement with controller makes me feel dizzy because its too quick and made the legs feel wobbly
Speed and ease of use of teleporting was very good to use
Not as unstable, better balance and easy to use
Teleportation doesn't make me immersed in the game, it just feels like I'm always standing. Swinging with arms - a lot of effort, arms started hurting, have to multitask. With Vehicle - fun, made me a bit sick, best way to move
Movement via arms felt realistic and more immersive. Teleport is quick and efficient but doesn't feel realistic. Smooth movement screen drag wasn't immersive, without vehicle made me feel wobbly.

Were any of the movement methods difficult to use, in context of playing the game? If so, why?

10 responses

yes. the vehicle one because if I was to turn, I would also have to make the box turn with me.
Teleportation to a bit of time to get used to, but was really good after a few trials. Swinging arms made if difficult to move and fire since i used both arms to move. Vehicle occluded the scene making it hard to see some opponents.
Swinging arms could be difficult to traverse long distances, but teleportation made it harder to escape out of negative situations
number 3 moved too much when moving away from enemies because it was very sensitive
The smooth movement because you had to move the box to face a different way and go forward at the same time.
one with the vehicle because the turning of the robot is confusing
None were difficult to use but vehicle movement felt annoying
The last method was a little difficult due to multi tasking, took a while to get used to
using swinging motions - just too much effort, arms started to hurt and I wasn't focused on shooting people. Maybe would be good in a skiing game.
Teleporting into a wall made me disorientated which made finding my way round harder.

Rank the movement methods from 1 – 3, where 1 is the type that felt the most comfortable using, and 3 the least comfortable using:



Why did you rank the movements as you did?

10 responses

I liked moving smoothly, and was the most comfortable. But the vehicle was very very frustrating. Teleportation jolted me about a bit and felt uncom,fortable, partticlualry for my head. Movement with swinging arms was a bit tiring but otherwise fantastic.
Teleportation easiest to learn and use, smooth the movement is easy but rotation is difficult. Swinging arms was a pain to use
Same reason as above
number 3 was the worst because it was hard to control movement speed. number 1 was teleportation because it was very easy to aim the arc to where i wanted to teleport
As above.
teleportation is the most comfortable because it is quick and smooth whereas the vehicle one feels too close
Smooth movement felt less intuitive in the context of the game
First method was easy and got to where you wanted quickly, whereas the others took longer and was harder to navigate
I really liked moving without the robot but it made me really sick, swinging arms made me feel tired, teleportation was comfy like a cat's butt in a box
swing arms felt like I was walking normally, teleporting felt a bit jerky. Smooth movement didn't feel that natural.

What changes would you make to any of the movements to suit them better to yourself?

10 responses

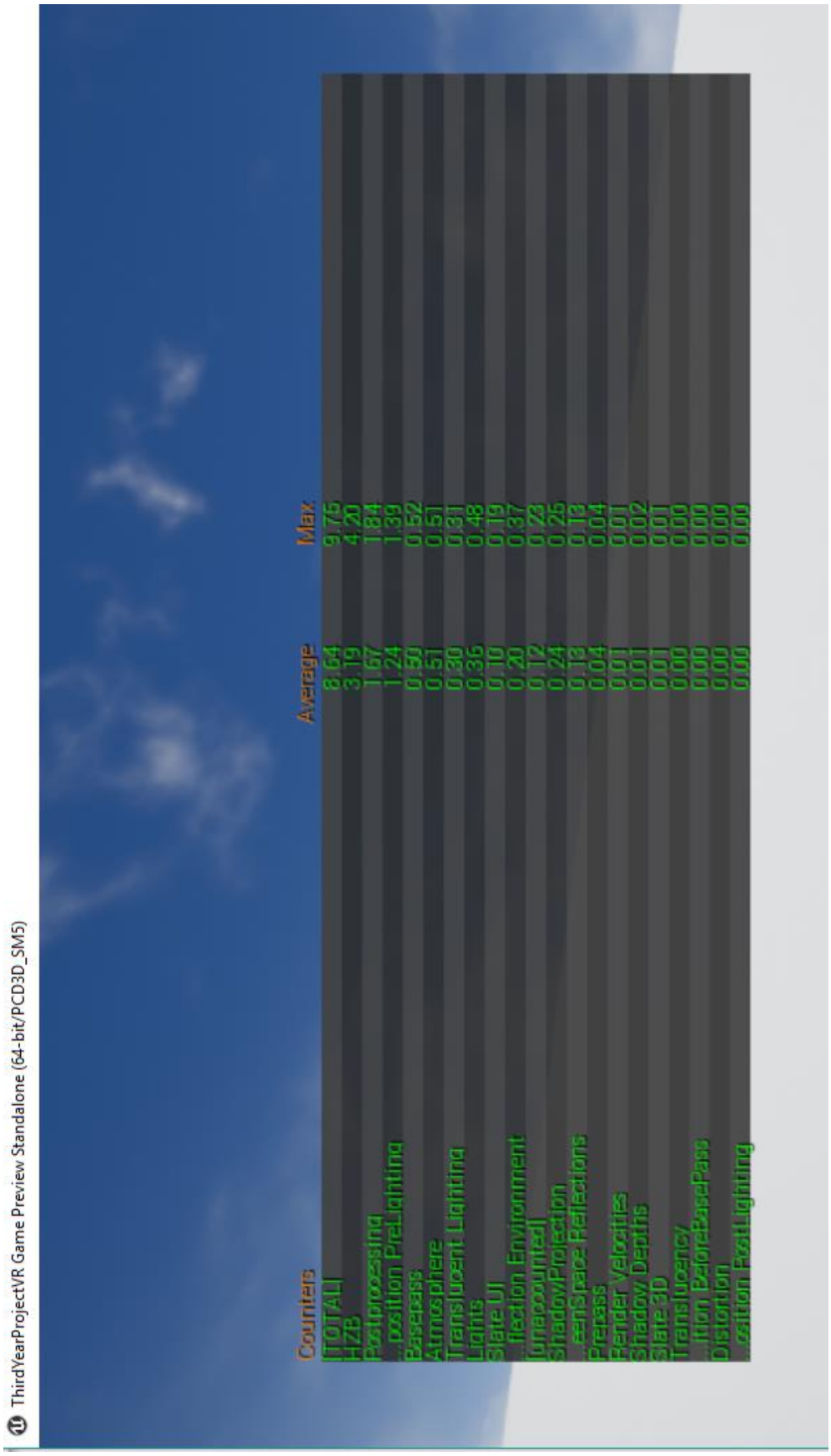
smooth movement without the vehicle ty ty. teleporting for very very short adn long ranges of dfistance.
Merge smooth and teleportation (too be able to got smooth movement and move far distances)
Make swinging arms have a greater than 1:1 ratio so you move faster
for the swinging arms one, leg trackers would be nice to use especially to make it more realistic than it already was. leg trackers would make it more fun. for the smooth movement, some sort of change to the speed where you were able to control how fast the character moves
Make it so that you can turn and move forward at the same time.
the colour - it is too white and bright
Faster movement in swinging arms, teleportation having some sort of animation so that it wasn't an instant snap to the next place, smooth movement with vehicle could have used a wider FOV
Second method to have longer movement strokes, giving you more control and doesn't take as long to move around
remove the robot, add some sort of indication to stop motion sickness; teleportation - use it for storytelling games, not for fps. swinging - make it faster
direction of vehicle in smooth movement moves with head, maybe swinging arms without drag.

Final comments, anything you'd like to say about the experience?

9 responses

:D i had a lot of fun!
Was fun to play and liked the idea. When trying the smooth motion without the box did make it much more harder (felt unbalanced and a bit sick). But with the box i didn't feel such motion sickness
10/10 would play again
it was very good. i enjoyed the game and would play it again. I would like to see more levels and harder difficulties. overall 10/10
It was a great experience and really fun.
Experience was fun
Really good research to explore the different movements in an VR game, and how to minimize sickness or feeling unwell whilst playing
I liked it, the biggest problems were - motion sickness and not feeling immersed; Well done on the project
Where is the 360 treadmill?

Appendix E – Black Box Test 8 Figure in Large



Appendix F – Original Project Brief

Technical demonstration exploring movement methods in Virtual Reality games - COMP3200 – Project Brief

Student Name: Imran Bepari Supervisor Name: David Millard

Problem

With the recent release of Virtual Reality head mounted displays on a consumer level, the gaming industry has still yet to properly grasp how a Virtual Reality (VR) game should be made on the scale of a Triple A title. Movement is a difficult subject in VR because of its boundaries. A user would be restricted to the room they are in, so walking around a large level or world the game to create a new movement system to make the game playable.

Many games will simply fit into the users play space and not move them around, or use a mechanic such as teleporting to move the player. While these work for some game genres, such as games as short indie games, a solution has not been found for large scale games with huge world, such as RPGs and shooters.

For larger games, a teleport mechanic is an unsatisfactory solution, and would not work for the intended user experience. Multiplayer shooter combat is broken if the player can teleport around, players need to be able to be seen to be able to be shot, for example. Teleporting can also thematically be unfit for a game, for example, if a game has a medieval theme, why can players move like that? Overall, a new solution is required.

Goals

Explore the different movement methods available in Virtual Reality, and discover how they affect the user experience in the game. These methods should be convenient for the user and not cause any motion sickness, while widening the scope of what VR games are capable of.

I will create three separate movement techniques in Unreal Engine 4, and try to compare them. The main goal of these demonstrations is to have large worlds and levels in mind with the movement design. These would all be incorporated into the same game. They can then be compared to see what kind of user experience they would create, and what kind of games could be created with them. The movement techniques would include:

1) the teleportation method, implemented in it’s best form for user experience.
2) the user mimicking a walking motion in some way to create movement in the Virtual world.

3) adding the user to a vehicle of some variety to give “context” to their location.

Scope

The scope of my project would involve a game tech demo that would have the three different movement techniques available.

This game would not be large in the sense of having a deep structure and length, but have a couple of large levels that require the player to move around a lot, in order to make full use of the systems being developed.

No assets or art will be created, mainly placeholders and royalty free ones so focus is on the games system.