# Final Report

## Title Page

## Acknowledgements

## Abstract

Contents

[Final Report 1](#_Toc480380239)

[Title Page 1](#_Toc480380240)

[Acknowledgements 2](#_Toc480380241)

[Abstract 2](#_Toc480380242)

[Main body 4](#_Toc480380243)

[Introduction 4](#_Toc480380244)

[Background and Project Objectives 4](#_Toc480380245)

[Personal Objectives 6](#_Toc480380246)

[Deliverables 6](#_Toc480380247)

[Literature review (if applicable) 6](#_Toc480380248)

[Project Management / Method of approach 6](#_Toc480380249)

[Legal, social, ethical, and professional issues 11](#_Toc480380250)

[Design 11](#_Toc480380251)

[Implementation 13](#_Toc480380252)

[Stage 1 15](#_Toc480380253)

[Stage 2 15](#_Toc480380254)

[Stage 3 15](#_Toc480380255)

[Stage 4 15](#_Toc480380256)

[Project post-mortem 15](#_Toc480380257)

[Conclusions 16](#_Toc480380258)

[Statement of word count 16](#_Toc480380259)

[Reference List 16](#_Toc480380260)

[Bibliography 16](#_Toc480380261)

[Appendices 16](#_Toc480380262)

[User Guide 16](#_Toc480380263)

[Project Management Artefacts 16](#_Toc480380264)

[Other materials (UMLs, designs, test results, sounds clip URLs) 16](#_Toc480380265)

## Main body

### Introduction

The purpose of this project was to create a Virtual Reality game made using Unity3D, intended for use with the HTC Vive HMD (Head mounted display).

Although there was never a definitive intended client, there are many potential ones. Small innovative indie games can often end up on digital distribution platforms such as Steam after being noticed by online communities such as Steam Greenlight or itch.io. In this case the intended client is any consumer interested in pc gaming, particularly those who are interested in small-scale, innovative games.

### Background and Project Objectives

**Initial idea for the project**

The initial plan for the project was a single-player VR puzzle game. After exploring the best potential puzzle-like applications for VR, a plan was drawn up for a “Factory Worker Simulation game” . The player would stand in front of a conveyor belt with different items moving past them on the conveyor. The player would have to do something with the items e.g. sort them, throw specific ones away etc. However, after about a week this idea was scrapped. (An image of the prototype can be found in the appendices). The reason for this was because the idea could not easily be expanded on to produce a full-bodied game. This was not a wasted experience though, as it highlighted that more planning was necessary to ensure that an idea be fully fleshed out before any development starts.

More planning let to a new idea – a game consisting of a set of “mini-games”. Inspiration was taken from Valve’s “The Lab”, described as a “compilation… of room-scale VR experiments” ("The Lab (Video Game)"). With high-end VR being (relatively speaking) in its infancy, these sorts of games give the best introductory experience to virtual-reality due to their varied nature and intuitive mechanics. The idea was that the player would start in a hub room and could travel to whichever mini-game he chose to play/experience, and then travel back to the hub room to select a new game.

To solidify the project’s objectives a Project Initiation Document (PID) was drafted (available in the appendices) which gave an overview of the project details. The aims for the project were specified in terms of its scope. This was split into things that should definitely be achieved, things likely to be achieved, and things that may be achieved if there is enough time. The PID proposed a “puzzle based game”, with the ability to “pick up and interact with objects”. It also specified at least “20” short puzzle based levels, 15 3D modelled assets and at least 10 hand recorded sounds. In hindsight, it appears that most of these aims were met, whilst some that ended up becoming no longer entirely relevant were re-adjusted.

Talk about choosing PlaySpace VR later and trying to make it look like a play room

**Choosing a platform**

One of the big initial steps in planning the project was to choose what platform it would be built for. I had to decide between a mobile or non-mobile platform. In the last decade there has been a “rise of smartphones with high-density displays and 3D graphics capabilities” ("History Of Virtual Reality - Virtual Reality"). This makes them extremely practical as virtual reality devices. Google have already taken advantage of this by launching the ‘Google Cardboard’, bringing “immersive experiences to everyone in a simple and affordable way” ("Get Cardboard – Google VR"). Samsung followed suit soon after by releasing the ‘Gear VR’ ("Samsung Gear VR With Controller"). Both these products are simply devices to hold your phone comfortably in front of your eyes. However, they simply do not have the processing capabilities to provide a smooth experience in most cases.



Figure 1 - Google's Cardboard headset and Samsungs Gear VR

In addition to the surge in cheap, mobile VR devices and applications, there has also been huge growth in VR head-mounted displays (HMDs). Three main competitors corner this market at present: Sony with the PlayStation VR, Oculus (now owned by Facebook) with the Oculus Rift, and HTC with the HTC Vive. These devices offer unparalleled performance for VR experiences due mainly to the quality of the displays. For example, both the Rift and Vive offer “two OLED panels boasting a combined 2,160x1200.” This means that “each eye gets its own 1080 x 1200 display” (Swider). As the goal of VR is to convince the brain that you really are in the virtual space being simulated, it is essential to have displays that offer a resolution as close to that of the human eye as possible. At present this is the main drawback of mobile VR; it simply does not offer a good enough resolution to make the experience completely convincing or immersive. After conducting this research, I settled on a non-mobile HMD as my target platform – specifically the HTC Vive because at the time only the HTC Vive had support for Unity3D, via an asset store plug-in.

### Personal Objectives

The personal objectives whilst undertaking this project were to develop skills relating to Game development. These include but are not limited to Game Design, asset creation/3D modelling and proficiency in specific software such as Unity3D and Blender. The developer also aimed to improve his knowledge and skills in developing for VR. In addition, carrying out a project of this scale as a solo developer from start to finish provided invaluable experience in development methodologies, effective versioning using GIT, bug tracking and time management.

A further objective was to gain experience in marketing a game, through producing a trailer to listing it on a sellers platform such as Steam or itch.io.

### Deliverables

The deliverables were a series of mini-games built in Unity3D for the HTC Vive. The games should offer a good diversity of experience in terms of VR interaction. They should also be intuitive and appropriate for all ages.

The main deliverable for this project was a Unity executable. However, I also gathered requirements information, produced a Gantt Chart to plan my work, took screenshots and video footage throughout development and drew designs at the start of the process. Together these provide a clear story of how the project progressed.

### Literature review (if applicable)

### Project Management / Method of approach

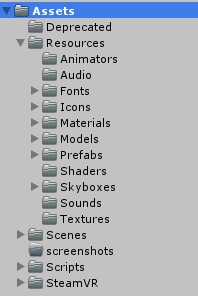
During this project, several software tools were utilised. The main three were Unity (for game development), Blender (for asset creation/modelling), and Git (for versioning).

A good, well-thought out approach can save a huge amount of time during a large project. Unity, Blender and Git all have ways to speed up and optimise the development process, especially when iterating between all three. For example, the developer set up Blender to save all assets straight into the Unity asset directory (under a subdirectory named “models”). This meant that when a blender file was saved, it would automatically update the Unity asset, keeping everything in-sync.

**Unity**

In terms of Software, Unity was chosen for the game development. This was down to several reasons. Firstly, the developer already had experience creating other, smaller, games with it. Secondly, at the time of the start of the project, the only other viable alternative (Unreal Engine), did not yet have any VR support.

Unity was for the most part a pleasure to work with. It provides organisational and structural tools which ensure as the project grows it does not become overwhelming or confusing, provided you put in the effort to keep things organised. For example, organising all assets into a hierarchical format can keep things streamlined and easy to navigate.



Another benefit of Unity is access to the asset store, where you can download third party assets. As the developer wanted to keep the project as independent as possible he refrained from using this, but it did provide the SteamVR Unity plug-in and the Skyboxes that were used in the project.

The SteamVR plug-in for Unity was a compulsory asset if the project was to be targeted at the HTC Vive. The plug-in allows developers to “target a single interface that will work with all major virtual reality headsets from seated to room scale experiences”. It also provides access to the tracked controllers and a useful 2D debug mode so the developer can move around the project scene using the keyboard if needed instead of having to put on the headset which is a useful time-saver.

Unity also makes it very easy to deploy to multiple platforms, which means that if the intended platform changed at any point in the development process it would make this transition much more manageable than other development tools.

Within Unity I set my default external scripting tool as MonoDevelop. MonoDevelop is a lightweight IDE used mainly for scripting. It includes all the essential features such as automatic code completion, source control and a GUI. However, the developer had problems with MonoDevelop crashing halfway through development and eventually switched to Visual Studio, which soon proved to be a more robust and stable IDE.

**C#**

Unity has only ever supported three languages – UnityScript (AKA JavaScript for Unity), Boo, and C# ("Documentation, Unity Scripting Languages And You – Unity Blog"). In 2014 only 0.44% of Unity developers used Boo, whilst 80.4% used C#. Since the Unity5.0 release Unity dropped support for Boo documentation to focus their resources in a more constructive way.

Due to user feedback they received in 2014, since then they have been aiming to provide in their documentation “C# examples across the board” whilst moving internally to provide the “best support for C#” that they can. It was decided, from this research, that C# is the best supported and most popular language to use in Unity 5.0, and should be the language of choice for this project

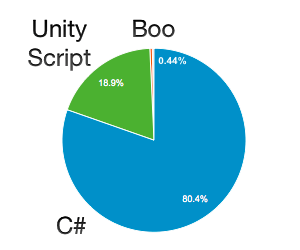


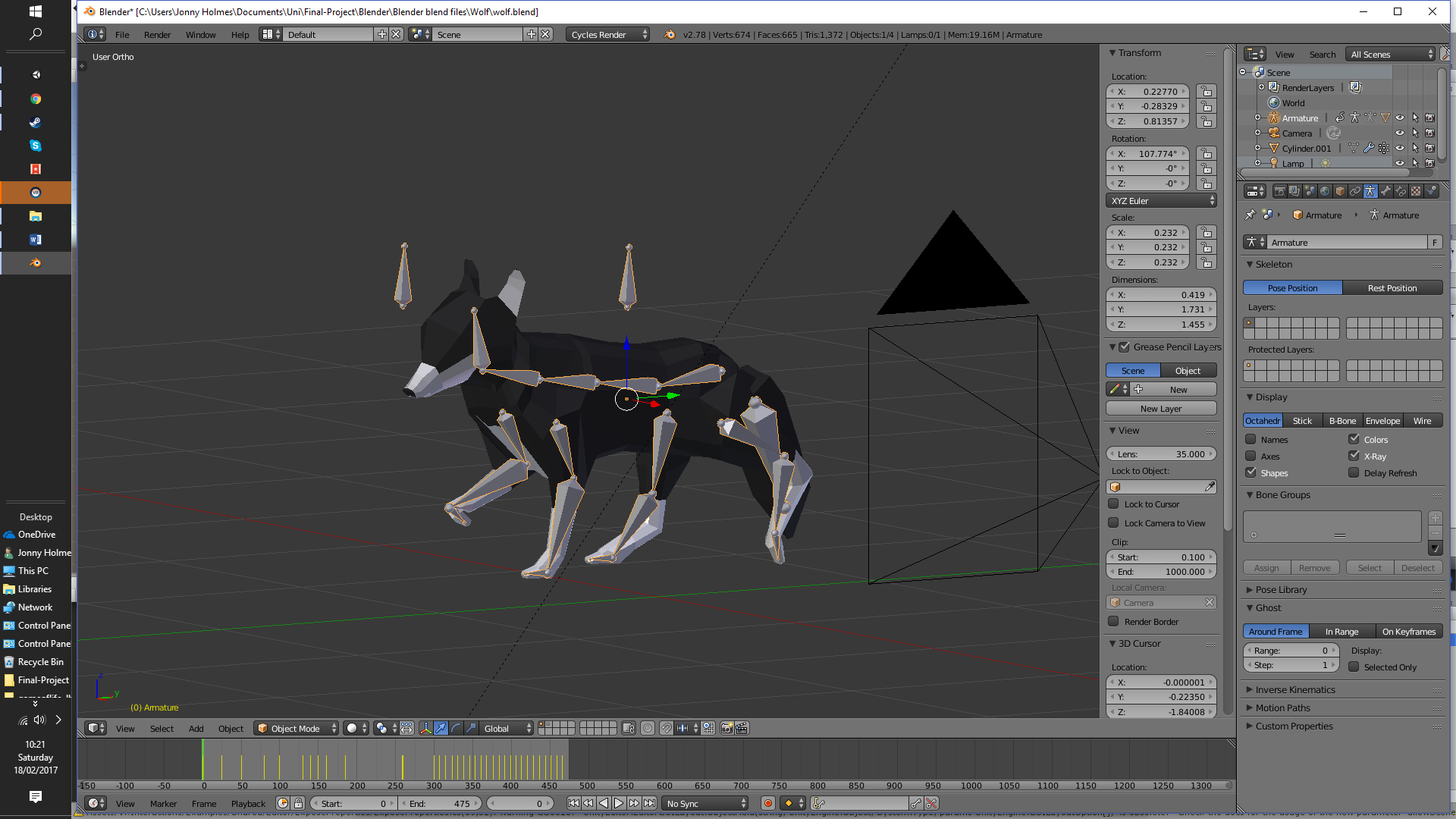
Figure 2 - Unity's official blog showing the proportion of each language used in 2014

**Blender**

Blender was used for modelling / asset creation. Although Blender can be intimidating or seem chaotic to some developers, when focusing on a particular set of features the workflow becomes quicker and easier over time. It provided the developer with a diverse set of tools to model, rig and animate as he needed.

**Otomata**

To create music for the game, I used ‘Otomata’, a “Generative Musical Sequencer”. It “employs a cellular automaton type logic… to produce sound events” ("Otomata - Generative Musical Sequencer"). Otomata was used to randomly generate sound patterns which were then used as music tracks for the game levels. It proved an intuitive and easy to use tool which produced an unlimited number of note sequences. As the synthesiser used for each note is the same it also provided a consistent sound across all the levels of the game. The licensing rights and requirements for this tool are discussed in the ‘Legal, social, ethical and professional issues’ section of this report.



**Version control**

Git was used throughout the project as the version control software of choice. Instead of using the standalone GitHub software, GitBash was used to track, commit and push files from the local to the remote repository.

A commit was made almost daily during work periods, however when the project work required balancing with other commitments this number dropped to once every few days. This frequent backup ensured that if necessary the project could be rolled back to a previous commit, reducing risk of data loss through hardware failure or other means. Roll-backs were done on multiple occasions and potentially saved the project days of work. For example, when updating the version of the SteamVR plug-in for Unity, some bugs became present, probably because the update was new and not completely stable. Even after rolling back the plug-in version the bugs remained. It seemed the only solution was to revert back to a previous commit and wait for the bugs to be fixed before attempting the update again. Sure enough, after updating a week or so later no bugs emerged.

A few problems arose with Git throughout the project duration, which were documented in highlight reviews. One problem occurred because GitHub will not allow free users to upload any file over 100MB. Some of the project’s Blender files were above the limit. This problem was solved in two ways. Firstly, by compressing the Blender files by default to reduce their size; secondly by adding the Unity project metadata folder to the repository’s ‘gitignore’ list. This did not reduce the integrity of the backup as this meta-data is generated every time the project is launched in Unity and so can be excluded from versioning.

**Development Methodologies**

As software development processes work much better with a team rather than individual development, a slightly laxer approach was adopted during the project. A toned-down version of XP (‘Extreme Programming’) was followed. The PID laid down a group of processes that should be followed, but soon after project start only the most effective ones were followed strictly. The XP processes followed were –

1. Keep a spreadsheet of required features, prioritized.
2. Define specific engineering tasks to get done (This ended up being done on paper as there were many small tasks could be dealt with quickly) A notepad was always kept at hand.
3. Time-box each session. (This was done for most (but not all) sessions). It did become clear that on average more work was done during time-boxed sessions, and it made it easier to quantify exactly how much had been done in a specific amount of time.
4. Utilise effective versioning (Using git as discussed).
5. Frequently reprioritize Gantt spreadsheet appropriately.

The most helpful of these processes was keeping a Gantt chart to organise the project over time. The chart made it easy to reprioritise tasks and their expected work days of effort and shift deadlines along if priorities changed. It also provided a way to know if the project was broadly on track at any point. This allowed the developer to spend more time on extra features for certain areas of the game if he was ahead of schedule or vice versa if he was behind schedule. The chart is split into rows denoting each task, sub-tasks, their start and end date and the number of work days predicted to complete them.

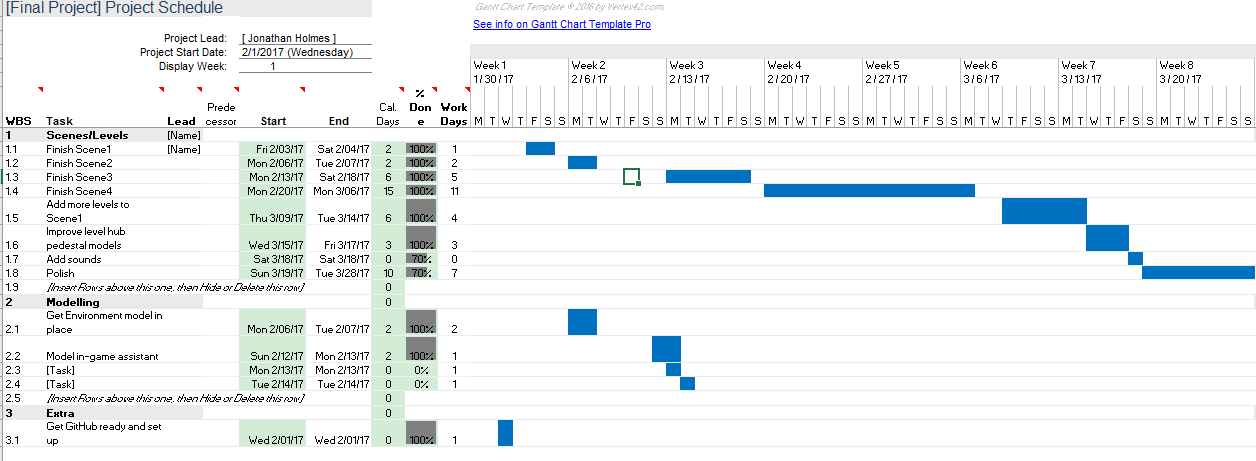


Figure 3 - Gantt chart used to organise the project over time

**User feedback**

Feedback was obtained using two main methods. Firstly, throughout the project the game was tested by 2-3 different people and feedback was received. A greater breadth of feedback could have been obtained by testing with even more people, preferably those of different ages, however there was no feasible way to carry this out. Many improvements were made to the project based on this user feedback. For example, at one point in the project the user playing the game would have to press the menu button on the Vive controller to return back to the level hub. However, one tester made clear that having too many controls to remember can become confusing and frustrating in VR as you cannot see your hands directly with the headset on. Instead he suggested the user interact e.g. grab an object in the game level to return to the hub (the same method used to get from the level hub to a game). This way the user never has to fumble with the controller and intuitively knows what to do.

The second, more diverse way of obtaining user feedback was through the internet forum 'Reddit'. The developer made a submission to the HTC Vive subreddit showing a clip of some gameplay and a request for any thoughts / feedback. A good amount of feedback was obtained from a total of 24 comments.

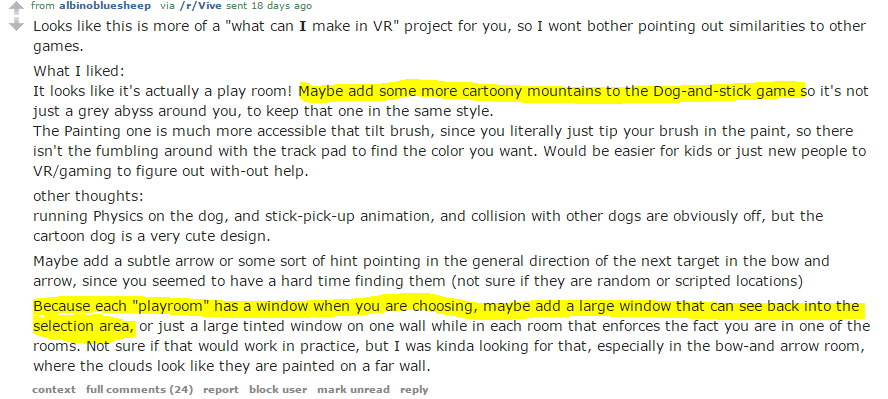
****

Figure 4 - A reply to the developer's post on the HTC Vive sub-reddit

The above figure shows an example feedback comment, in which the user says what they liked and didn’t like about the game, and suggests some improvements which have been highlighted in yellow. The first suggestion is the add ‘cartoony mountains to the Dog-and-stick game’. This was a valid suggestion as at the time the environment just faded into fog. The next suggestion was to be able to see back into the ‘selection area’ or level hub from each game area. This was another good suggestion as it provided some consistency to the game and at the same time made it seem more like a series of play rooms. Both suggestions were implemented in the next sprint session.

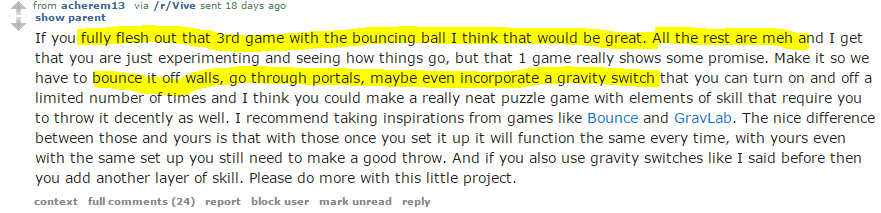


Figure 5 - A reply to the developer's post on the HTC Vive sub-reddit

Another comment suggested that whilst one of the 4 mini-games showed promise, the others were lacking. The writer also recommended fleshing out further the ‘ball-in-pipe’ game to make it more interesting. Due to time constraints, this feedback was not implemented. However, these features could well be implemented in future as I plan to continue developing the project after the deadline and so their value is still great.

**User Stories**

As part of the project’s agile approach, user stories were drawn up summarising the desired gameplay functionality. To draw these up the developer played VR games already on the market and took note of what he deemed essential for a comfortable, immersive VR experience. These user stories also offer good basic criteria to conclude whether the final product met it’s aims.

### Legal, social, ethical, and professional issues

**Legal**

The main 3rd party asset used (the SteamVR sdk / plug-in ) was free to use for commercial use.

All music generated using Otomata, the generative music sequencer, is open for commercial use, however the creator states that he would appreciate some attribution, but it is not compulsory.

All sounds downloaded from ‘freesound.org’ are under the Creative Commons 0 license, which means any user can “copy, modify, distribute and perform the work, even for commercial purposes, all without asking permission”("Creative Commons — CC0 1.0 Universal").

For the trailer the developer produced to showcase the game, an audio track was used from ‘Bensound.com’ ("Royalty Free Music By Bensound | Creative Commons Music"), and appropriate attribution was given.

**Professional**

**Coding standards eg. DRY coding**

A matter of high importance during this project was to keep the code neat, readable and consistent. This was essential when such a large number of scripts were involved. Principles such as DRY (Don’t Repeat Yourself) were enforced from the start. It stayed an important goal that every script and indeed every function had ‘a single, unambiguous, authoritative representation within a system’ ("Don't Repeat Yourself"). This meant that even after weeks or months of not working on a area of code, it was easy to come back and continue on it without becoming confused. In Unity3D’s object based system, this was achieved through attaching multiple scripts to one object, each with its own clear behaviour. E.g. for the game mascot (a cube character who flies around exploring the levels), one script attached handles the movement whilst another script is attached to the thrusters whose sole purpose is to keep the thrusters pointing towards the ground. This modular approach keeps the scripts organised and intuitive based on their name and the object they are attached to.

It was also essential to stick to consistent coding conventions in terms of line-spacing and commenting. The developer adhered to the Unify Community C# Coding Guidelines ("Csharp Coding Guidelines - Unify Community Wiki"), which specify correct indentation for loops, correct use of bracing, comment style and spacing. For example, the document specifies that comment tags should be placed “wherever possible… above the code instead of beside it”, and that “braces should never be considered optional”. Having solid conventions such as these saves time not only when writing the code but also when reading it back.

**Ethical**

### Design

**Level design**

During the project, sketches and designs were continuously drawn up, along with the desired basic functionality of the game. This allowed initial ideas to develop and change early on, ensuring a solid idea before any modelling or programming began. From here an iterative approach was adopted to achieve the desired level design in Unity.

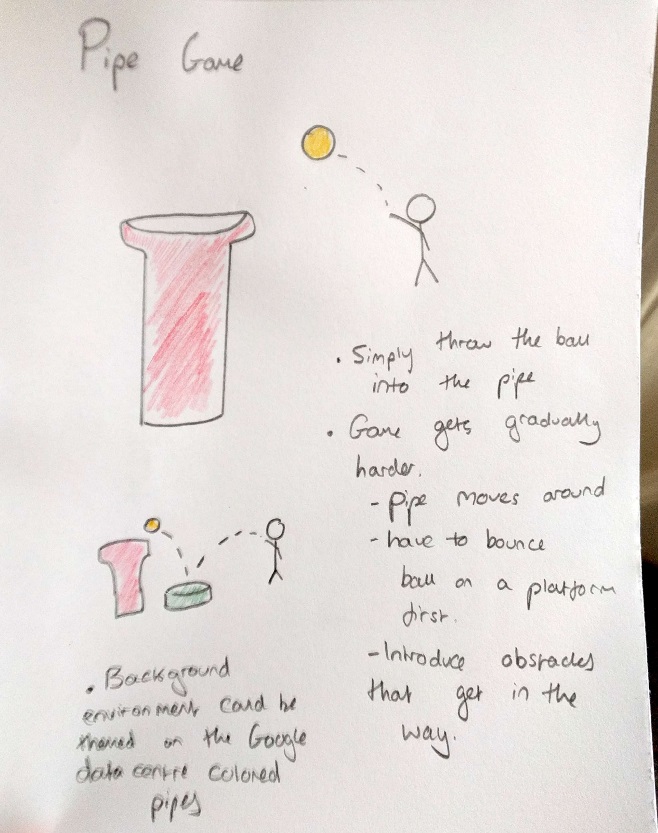


Figure 6 - Level design sketch made for the 'Ball-In-Pipe' mini-game

Once the idea for the game had been solidified, the developer would then experiment in Blender to scope out how the assets would best be modelled. For the ‘Ball-In-Pipe’ mini game, for example, it was planned that the player would be able to see a series of interconnected pipes outside the play area, to give the feel of some sort of factory. In this example, instead of drawing up a sketch, a reference image was used. The developer wanted it to look like Google’s famous datacentres and so used a relevant reference image to model against.

Once the required assets had been modelled, the level design process was then taken to Unity3d. As the blender files were always saved to the Unity assets folder, this meant that they could be iteratively edited in Blender and any changes would be automatically updated in Unity.

### Implementation

**Modelling and animation**

The process of producing the models needed from the initial designs was a gradual one. First a basic pipe model was created. This was then expanded on to produce more complex combinations of pipes. These were then duplicated and rotated to produce a seemingly random array of pipes.

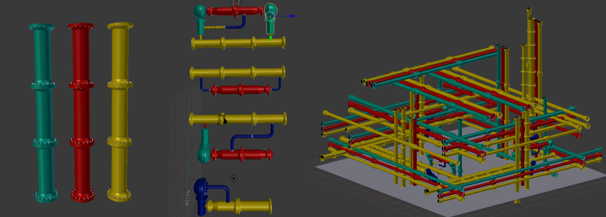


Figure 7 - The process of developing a model in Blender

This approach combined initial planning with experimentation until a result was achieved that fitted what the developer envisioned.

Blender’s armature system was utilised to allow for easy animation. A model can be fitted with ‘bones’ which then join to a particular part of the mesh. When the bone is moved, the mesh will move. Once a model was rigged with an armature, it was animated using Blender’s dope sheet window via keyframes.

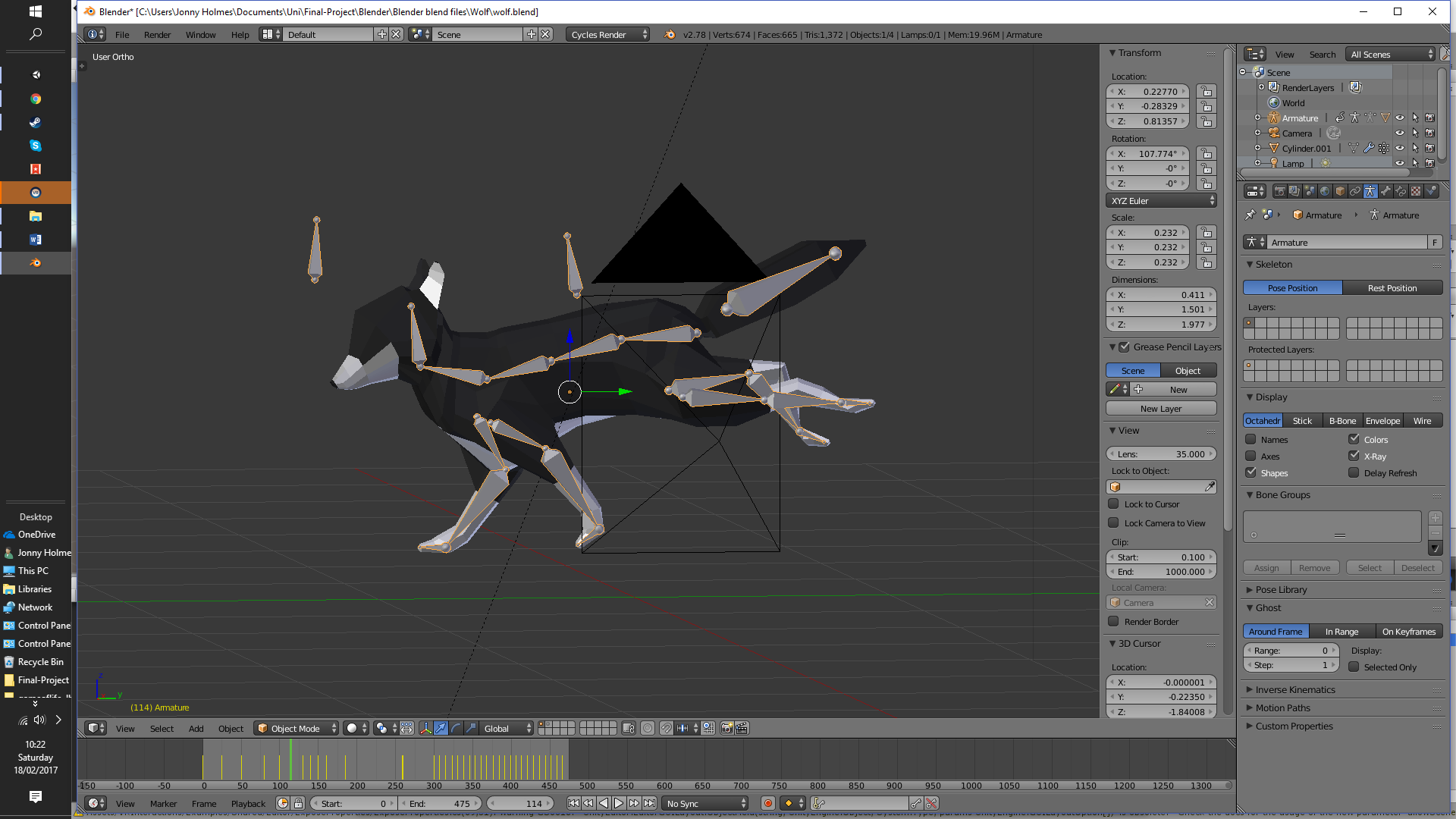


Figure 8 - Model rigged with armature, showing keyframes

The area of a mesh that a bone influences, along with the strength of the influence, can be changed in Blender’s weight paint mode. This enabled the developer to create various animations for a model with limited effort.

**Audio**

The required audio for the game was acquired using three methods. Some audio was created by the developer using a microphone and Foley methods, some was acquired from a website called ‘freesounds.org’ and some was created using ‘Otomata’, the generative music sequencer. Once the audio was ready in mp3 or wav form it was edited using Audacity. The purpose of the editing ranged from reducing/removing background noise, cropping unneeded sections or looping the audio. Once the audio has been edited it was then stored within the Unity assets directory under an ‘Audio’ subfolder. From there it was then used as needed by ‘Audio source’ components, which can be attached to Unity game objects to play music or sounds.

**UI**

For VR, UI is a controversial issue. Some developers will overlay UI elements directly onto the user’s camera view. However, most VR games adopt a system where the user interacts with virtual objects in the game world to replace this. The ‘menu’ system in PlaySpace VR simply consists of grabbing spheres with the trigger button on the controller. This makes for a much smoother experience than having to look at UI elements on the camera view or with a separate menu button on the controller.

**AI**

Several agents within the game have AI driven behaviour. The wolves featured in the ‘fetch game’, for example, utilise Unity’s navigation mesh system. This system allows the developer to specify game objects that are ‘navigation static’. This are objects which will never move and will form the baked navigation area for any agents that want to traverse it. Some wolves in the game will randomly wander by choosing random points on the navigation mesh and calculating a path towards it. The wolf which fetches the stick thrown by the player will calculate a path first to the stick, then back to the player, avoiding any obstacles in its way.

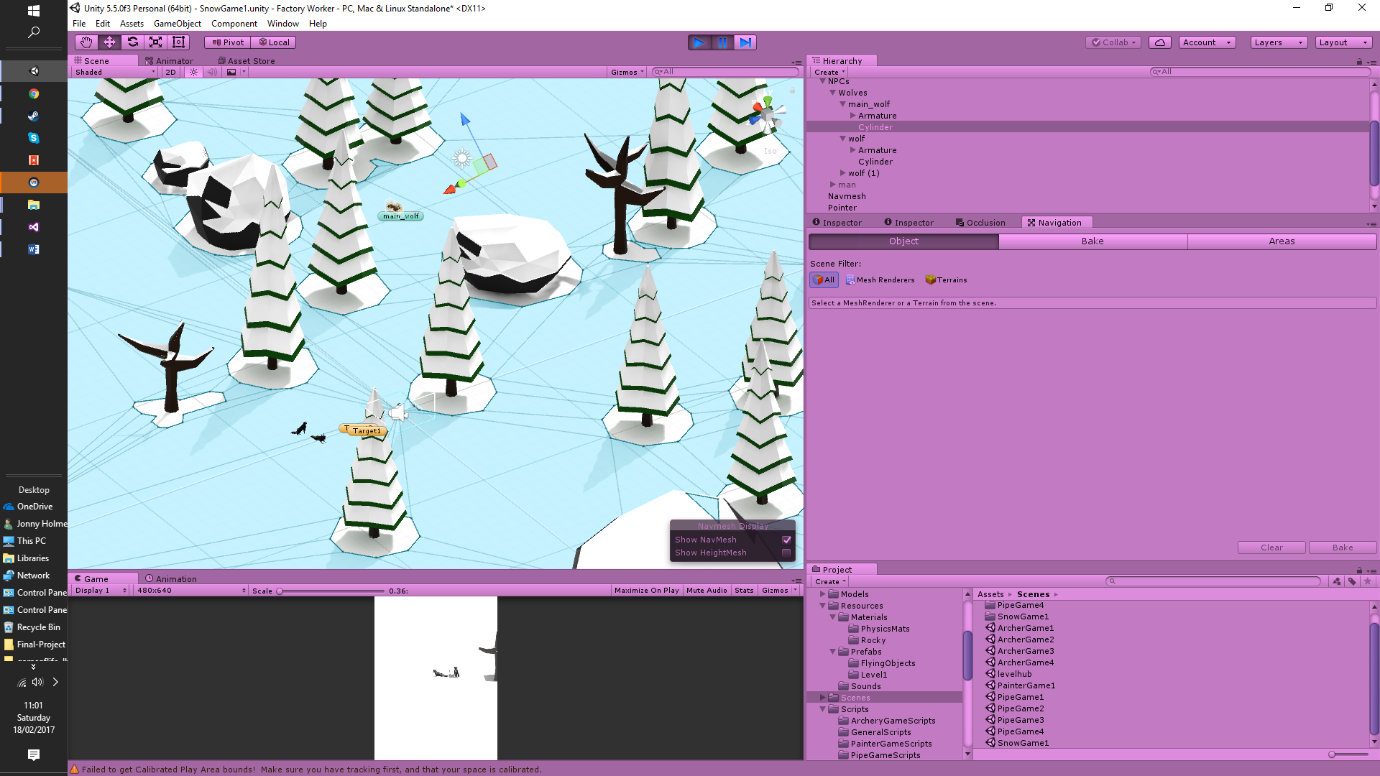


Figure 9 - Navigation mesh (in blue) in Unity

Another example of AI is shown by the cats that float around in the ‘painter game’. They follow an artificial flocking algorithm to move around. The algorithm was originally defined by Craig Reynolds and was known as the ‘Boids’ algorithm ("Boids (Flocks, Herds, And Schools: A Distributed Behavioural model)"). Each cat follows three rules: alignment, separation and cohesion. Alignment steers the heading of each Boid to the average heading of neighbouring Boids. Separation steers each Boid away from any Boid that gets to close. Cohesion steers each Boid to the average position of neighbouring position. Together they attempt to accurately emulate flocking behaviours of animals.

**Unity**

### Every Unity script inherits from the ‘MonoBehaviour’ class (Technologies). This class provides the functionality to start co-routines (effectively threads), which is a very useful feature. ‘MonoBehaviour’ itself inherits from the ‘Behaviour’ class, which provides the methods to enable and disable the script. Behaviour inherits from Component, which provides the base class for everything attached to ‘gameObjects’. This means not only user written scripts but also components provided by Unity. At the top is the Object class, which acts as the base class for all objects in Unity. It offers instantiation and destruction functions.

Creating and using
Object
Component
Behaviour
(enabled)
MonoBehaviour
(coroutines)
Script
 

Figure 10 - Unity's Class hierarchy (Studio)

### Stage 1

Most of Stage 1 was spent solidifying the high-level structure of the scenes. It was important to develop a stable prototype as soon as possible, as this would make it clear early if any part was not feasible or needed to be changed. The focus was on modelling the large environment scenes that were to be used in the archery and snow game. The wolf asset was also initially modelled, rigged and animated and a basic prototype of the snow scene was implemented where the player could play fetch with the wolf. However, at this stage the wolf would not look at the player and the model and texture was basic and rugged.

Basic implementations of the archery and painting scenes were also implemented at this stage. Only basic functionality was implemented but this helped to further solidify the structure of the game.

A further task that was done in this stage was to consider and design an in-game assistant. Designs were drawn up but nothing was implemented at this stage due to time constraints.

The version control that was to be used throughout the project was set up, and the Excel Gantt chart to be used as a plan was populated as far ahead as was feasible.

### Stage 2

Stage 2 was spent adding to the prototype created in Stage 1 to create a more substantial game. A level hub was constructed for the user to traverse the levels, and each mini-game was substantially flesh out. From this stage onwards these sections will be split based on the respective mini-games that were worked on during the stage.

**Painter Game**

A brush and an eraser were modelled and added to the painter scene. The logic of the line renderer code for the painter scene was implemented, allowing the user to draw lines with the controller. More wolves were added to the snow game which would roam around, making the world feel more alive.

**Ball-In-Pipe Game**

At this stage, this game was something completely different to what is in the final product. The first implementation was a game involving a conveyor belt and various shapes. The player had to move the shapes arriving on the conveyor belt into the right pipes. This was later replaced with the ‘Ball-in-pipe’ game visible in the final product.

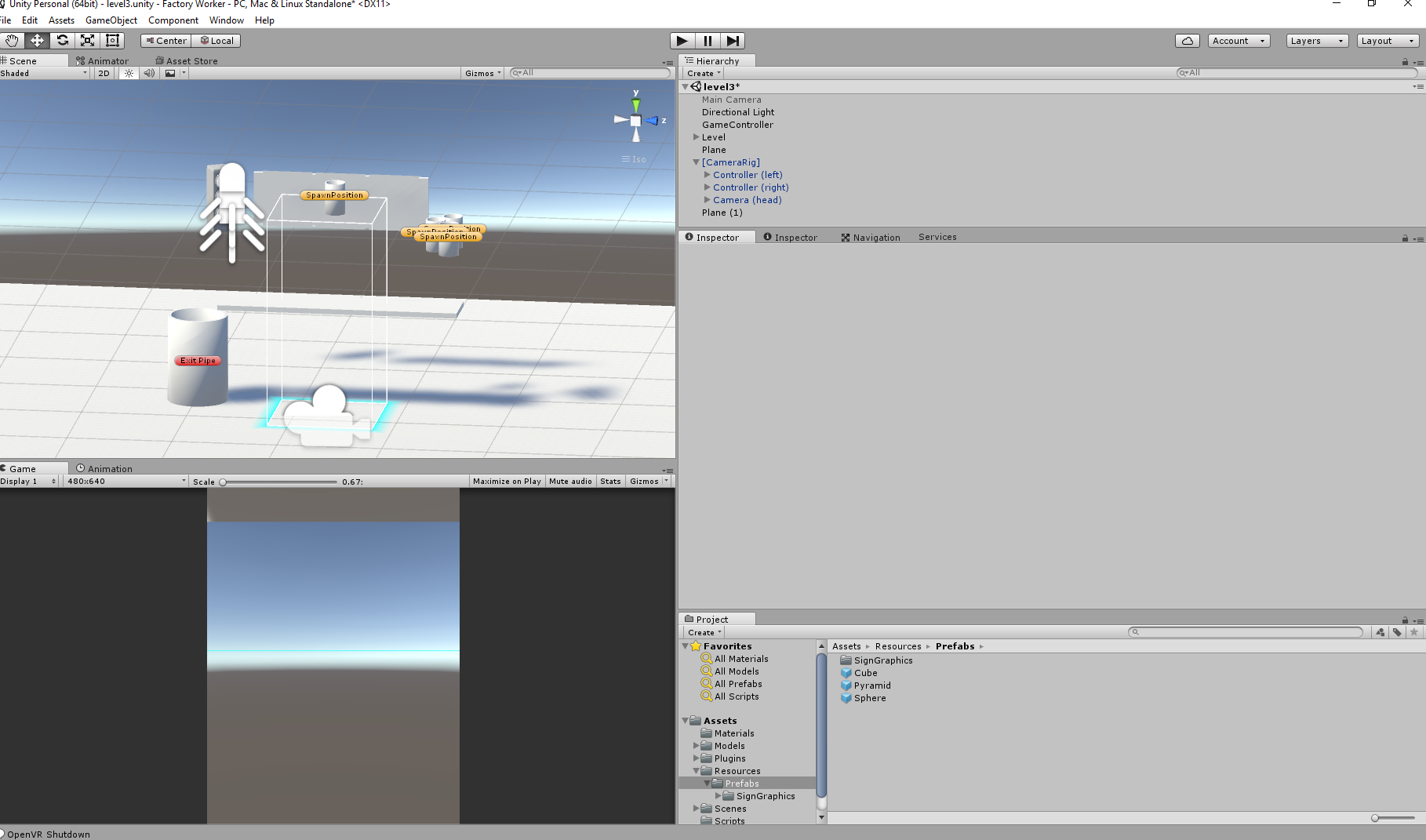


Figure 11 - Initial game idea which was scrapped and replaced

**Archery Game**

At this stage, the archery game was also unrecognisable to the final version. The initial idea was for an in-door archery range where the player could see all the targets at once and had to shoot all of them to pass the level.

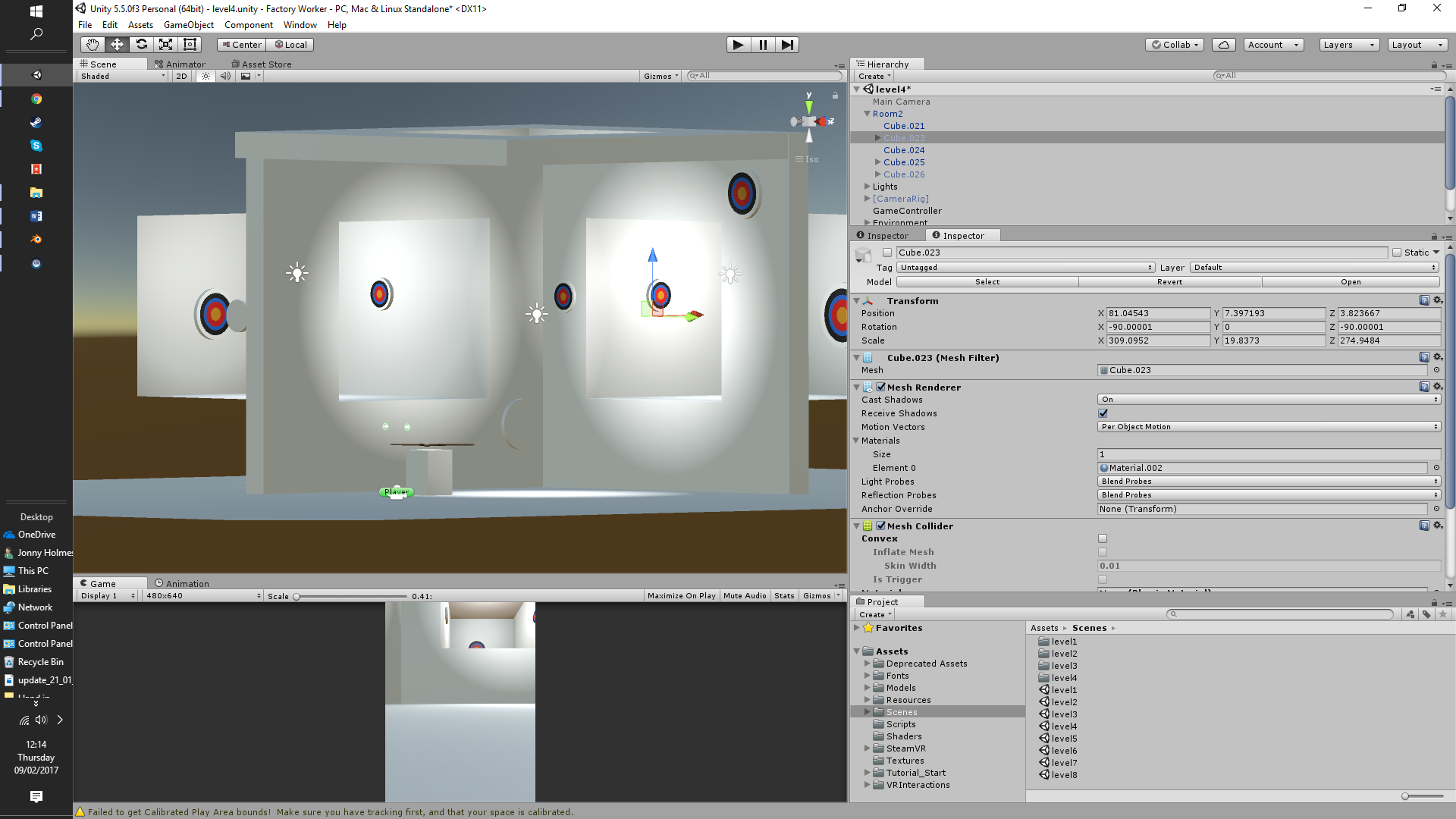


Figure 12 - Initial implementation of the archery game

Deciding to go back to the drawing board with 2 out of 4 of the planned mini-games highlights the importance of prior planning. Time could have been saved in the modelling and implementation phase simply by recognising early that there was a more effective solution available.

### Stage 3

**Archery Game**

The archery game was changed to an outdoor environment and its core game play mechanic changed from “hit all the targets” to “see how many targets you can hit in a specific amount of time”. This change was made after some user testing and the mini-game being described as “lacking a game mechanic that actually holds your attention”.

**Painter Game**

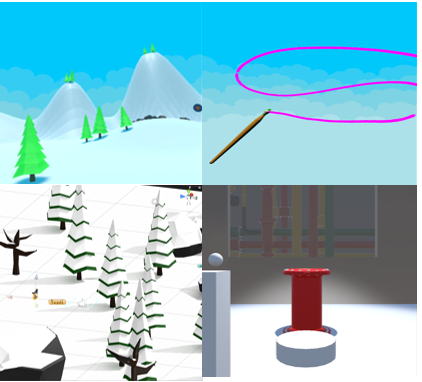
An important design choice was made at this stage for the painter game. At this stage, the user was selecting the colour that they wanted to paint with by pointing at a colour pallet with one controller and painting with the other. After some user testing it was clear that this was “fiddly” to use and detracted from the intuitiveness of the rest of the game. In addition, the future plan for the game was for one hand to be holding an eraser and the other a brush. It would be even more impractical to have the eraser hand functioning both as an eraser and as a pointer to the colour pallet. A decision was made to replace this colour pallet with a system of multiple paint pots filled with different colours of paints. The user would dip the paint brush in the paint colour that they wanted to use. The upside of this system was that it was much more intuitive and user-friendly, the downside is that the number of colours available was limited to the number of paint pots, whereas the colour pallet offered an infinite colour range.

**Ball-In-Pipe Game**

The ball-in-pipe game by this point had been completely revamped, and involved a new mechanic where the user had to bounce a ball off a platform with the correct trajectory such that it landed in a red pipe. This was to be the foundation that the rest of the levels would build upon. Feedback was received that this was a fun and intuitive mechanic.

**Snow Game**

Some good work was done to the Snow Game by this stage. The user could now play fetch with an animated wolf.



### Stage 4

The first half of the final stage was spent adding more features to the mini-games. The second half was spent polishing and bug-fixing. By this point the game had come together well and there was time to add a feature that, under the scope list defined in the PID, was only to be added if the time was available. This was an in-game mascot or assistant.

He was an animated mascot that would fly around each scene being curious and looking at various objects. He provided a needed element of consistency to make all the games feel as though they were connected and part of the same whole.



### Project post-mortem

**Personal Objectives**

The main personal objectives for this project were to improve the developer’s skills relating to Game development, and to gain experience with VR as the target platform. The former objective has been met as the developer feels much more confident in using Unity3D, C# and Blender. However, he is by no means proficient and there is clearly much more to learn from each of these utilities. The latter has had partial success in that experience has only been gained targeting the HTC Vive. If possible the developer would have liked to make the game cross platform to support the Oculus Rift, but this was impractical in terms of resources and time.

The developer has also become more skilled at using versioning software such as GIT. However, this objective has had limited success as the developer didn’t fully utilise the capabilities of the technology. He did not gain experience in managing and merging different branches and throughout the whole project stuck to one branch. This limited the full array of benefits that versioning offered and the skills learned. In future projects, he will be sure to utilities these features.

Another personal objective was to improve time management skills. This objective has been met through practicing time-boxed sprints where feasible, however it could have been more beneficial if these sprints were more strictly timed and organised, as they ended up being loose guidelines.

Finally, the developer wanted to gain experience in marketing an app. This objective was fully met via the production of a trailer[[1]](#footnote-1) and the listing of the game on the site itch.io.[[2]](#footnote-2) The game has sold one copy as of writing this report.

**Deliverables**

The main deliverables met the desired expectations well. User feedback gained through Reddit indicated that they did offer a good diversity of experience in terms of VR experience and was appropriate for all ages. However, it also showed that several users weren’t content with some of the mini-games not having an active purpose and just being a relaxing environment (e.g. playing fetch with a wolf). If the project were to be done again it would be ensured that all mini-games had an active game mechanic like the ball-in-pipe game.

**Method of Approach**

**Methodology**

The form of ‘XP’ programming that was followed was a success as it ensured the project was kept organised and on-track. It also allowed the developer to keep track of project priorities. Without this form of approach the project would likely have become disorganised and fallen behind schedule.

**Technologies / Implementation**

Unity proved a pleasure to work with. It posed very few problems throughout the project and its organisational tools kept the project neat and professional. It also coupled well with the use of Blender and saving Blender files directly into the Unity asset directory, saving time and effort. Blender was very useful for asset creation and animation. Other technologies such as Maya and 3ds Max would have been viable alternatives and posed no difficulties, but the developer was already familiar with Blender, and so time was saved by choosing it. C# posed no problems that couldn’t be solved via quick internet research or the Unity documentation, as the documentation now provides C# examples for almost everything. Choosing Boo or UnityScript would have been possible but would not have been a logical choice as the documentation for these is almost nil.

In terms of music creation for the game. Otomata proved a simple and easy way of procedurally generating music. However, it was limited by having only one sample sound. This gave all music generated a consistent feel to it, but ideally the music would have been better had it been more varied. If the project was done again a music creation tool such as FL Studio would be used to create some more interesting and professional music.

Git proved a good version control software, but did pose some problems during the project. For projects which can include slightly large audio or fbx files, it’s 100MB per file limit proved frustrating. This limit can be upgraded for a fee, however. This was the only frustration with it, and so Git would still be used in projects going forward.

**Initial planning (PID, user stories, designs)**

The requirements and aims laid out by the PID were mostly met. Of those that were not, this was usually due to the project changing direction over time, making these objectives not entirely relevant. For example, one “definite” aim was that the game should be “puzzle-based”. This only ended up being true in one of the mini-games (the ball-in-pipe game). Despite this, the scope laid out by the PID was a useful tool to allow effective prioritisation by specifying fixed numbers e.g. “At least 15 3D modelled assets”. This gave the project more solid requirements to aim for.

Obtaining user feedback through internet forums such as Reddit proved extremely useful. Targeting a forum that is based around the HTC Vive meant that most people who viewed the post would have a HTC Vive themselves and be interested in or even have experience playing VR games, making their suggestions even more important. Many changes were made to the project based on the feedback received. If the project was done again this would only be done more.

User stories were a small but extremely beneficial part of the planning process. They enabled the project to always keep in mind it’s most important aims in a clear way. If the project was done again, more user stories would have been written to cover a wider variety of user needs.

**Developer performance**

The developer found his performance throughout the project to be good all round but with room for improvement. He lacked the discipline to strictly follow a time-boxed time management methodology but at the same time put in the number of hours required to get the project done to the level that was desired. The developer is very pleased with the final product, especially in terms of its completeness, and feels that he has met almost all the goals that he set out with.

### Conclusions

## Statement of word count

## Reference List

## Bibliography

"Get Cardboard – Google VR". *Vr.google.com*. N.p., 2017. Web. 16 Apr. 2017.

"Google Cardboard – Official VR Headsets – Google Store". *Store.google.com*. N.p., 2017. Web. 3

"History Of Virtual Reality - Virtual Reality". *Virtual Reality*. N.p., 2017. Web. 5 May 2017.Apr.

"Samsung Gear VR With Controller". *The Official Samsung Galaxy Site*. N.p., 2017. Web. 11 Apr. 2017.

Swider, Matt. "HTC Vive Vs Oculus Rift: Which VR Headset Is Better?". *TechRadar*. N.p., 2017. Web. 2 May 2017.

"The Lab (Video Game)". *En.wikipedia.org*. N.p., 2017. Web. 2 May 2017.

"Creative Commons — CC0 1.0 Universal". *Creativecommons.org*. N.p., 2017. Web. 5 May 2017.

"Documentation, Unity Scripting Languages And You – Unity Blog". *Unity Technologies Blog*. N.p., 2017. Web. 1 Apr. 2017.

"Otomata - Generative Musical Sequencer". *Earslap*. N.p., 2017. Web. 11 Apr. 2017.

"Royalty Free Music By Bensound | Creative Commons Music". *Bensound.com*. N.p., 2017. Web. 5 May 2017.

"Boids (Flocks, Herds, And Schools: A Distributed Behavioralmodel)". *Red3d.com*. N.p., 2017. Web. 2 May 2017.

"Csharp Coding Guidelines - Unify Community Wiki". *Wiki.unity3d.com*. N.p., 2017. Web. 29 Apr. 2017.

"Don't Repeat Yourself". *En.wikipedia.org*. N.p., 2017. Web. 5 Apr. 2017.

Technologies, Unity. "Unity - Scripting API: Monobehaviour". *Docs.unity3d.com*. N.p., 2017. Web. 4 May 2017.

## Appendices

### User Guide

### Project Management Artefacts

### Other materials (UMLs, designs, test results, sounds clip URLs)

### 

1. https://www.youtube.com/watch?v=6GpEoGTDQng [↑](#footnote-ref-1)
2. https://happyadjustablespanners.itch.io/playspacevr [↑](#footnote-ref-2)