School of Computing
FACULTY OF ENGINEERING

UNIVERSITY OF LEEDS

Vive Virtual Reality Technology Demonstration

Khen Cruzat and Philip Nilsson

Submitted in accordance with the requirements for the degree of Computer Science

2016/2017

The candidate confirms that the following have been submitted.
<As an example>

Items	Format	Recipient(s) and Date
Final Report (2 copies)	Report	SSO (DD/MM/YY)
Final Report (digital)	Report	VLE (DD/MM/YY)
Project Code	GitHub Repository	Supervisor, Assessor (DD/MM/YY)
User Manual	Report Appendix	Client, Supervisor (DD/MM/YY)

Type of project: Software Product

The candidate confirms that the work submitted is their own and the appropriate credit has been given where reference has been made to the work of others.

I understand that failure to attribute material which is obtained from another source may be considered as plagiarism.

Khen Cruzat

Philip Nilsson

Summary

The problem that will be solved for this project will be that the School of Computing wanted a technical demonstration for Virtual Reality hardware. This project will be creating that software.

Acknowledgements

<The page should contain any acknowledgements to those who have assisted with your work. Where you have worked as part of a team, you should, where appropriate, reference to any contribution made by other to the project.>
Note that it is not acceptable to solicit assistance on 'proof reading' which is defined as the "the systematic checking and identification of errors in spelling, punctuation, grammar and sentence construction, formatting and layout in the test";
see http://www.leeds.ac.uk/gat/documents/policy/Proof-reading-policy.pdf.

Contents

Chapter 1

Introduction

1.1 Problem Statement

The goal of the project is to produce a technical demo for the HTC Vive to be used by the university to showcase development skills for virtual reality. This demo would be used in the School of Computing open days.

1.2 Client Background

The client for this project is the School of Computing in the University of Leeds.

1.3 Problem Background

The School of Computing wanted this project to be done as currently they own virtual reality hardware, however they do not have any software made by University of Leeds students to show to potential students. They are currently using software bought online in order to show the capabilities of the virtual reality hardware. They would prefer it if the software that they used is made by students from the School of Computing.

1.4 Project Aim

The aim of this project is to create a technical demo for the School of Computing, using the HTC Vive. This demo should appeal to prospective students, as well as appealing to people in the industry. This means that the project has to be both technical, for the industry, and interesting, for the prospective students.
To make it technical enough for the people in the industry, features have been added that are not trivial to implement in Unreal Engine 4.
To make it interesting for the prospective students, the demo has to have good gameplay and an interesting concept behind it.

1.5 Possible Demo Idea

One possible idea for the demo would be to combine the Towers of Hanoi with graph flow. These could be combined by having a generated landscape with a randomly generated graph on it, matching the flow of the terrain. This graph's edges would be rivers, or ditches with water running through them, and the nodes would be either river intersections or pools of water. This would be merged with the Towers of Hanoi by using the Towers of Hanoi system as a dam, to block off flow to a certain river, or by moving the disks you could control the amount of flow. This would work by having the disks stack upside down, with the smallest disk at the bottom. This is done in order to accommodate the shape of the ditch. The less disks that are blocking the river, then the more flow it would have.

The goal of this demo would be to keep all the plants at each node alive. The plants would be considered alive if they got the right amount of water. Too much water they would die and too little water they would die too.
This would be a possible demo idea as it implements several features that are non-trivial in Unreal Engine. Tasks are classified as non-trivial if they cannot be done in engine. It also demonstrates two aspects that are covered in the computer science course, which are the Towers of Hanoi, and Graph Flow.
These features are:

- Running water
- Water Collision
- Having the plants be affected by the amount of water
- Randomly generated river "graphs"
- Towers of Hanoi logic for flow control

The trivial tasks, that are done in-engine, would be:

- Generating terrain and landscapes
- Simple Gesture Controls
- Simple virtual reality gameplay (including teleport mechanic)
- Physics
- Flowers on the terrain

1.6 Deliverables

1. A link to the full code repository on GitHub
2. An instruction manual, detailing how to compile the code, the objectives of the technical demo, and how to control the technical demo
3. Project Report

The reasoning behind these deliverables are:
The code is needed so that the assessors can see what has been for the project, and this will show all the progress that has been made on the software over the course of the project, and how each feature was implemented. This will be on the version control site that is being used for the project, which is GitHub. The Version Control page will be delivered so that the software engineering project management side of the project can be assessed.
An instruction manual was decided on so that the assessors know how to compile the code properly, so that they can test the software, and it will also detail the controls and the objective behind the game.
The project report should be delivered as it provides insight into the inner workings of the project. It also shows the knowledge that the authors gained from doing the project.
These will be the only deliverables as they fully encompass all the work done during the project.

Chapter 2

Background Research

2.1 Virtual Reality

Virtual Reality is a technology that has been around since the early 19th century, although in a primitive form through the use of stereoscopic photos [12]. Stereoscopic photos work by using two photos that are taken of the same place but are slightly offset from each other, as can be seen in 2.1. This creates an illusion of depth for the person viewing the images, when viewed through a stereoscope. A stereoscope is a viewing device that only allows one eye to see one of the two images, so each eye sees a similar, yet different image, and this gives the illusion of depth. Stereoscopic vision is the same technology used in current Virtual Reality headsets although now the images are moving.

Figure 2.1: Example of a stereoscopic image.

Virtual reality platforms have been released aiming to provide an immersive experience to consumers. There are many varieties currently available and they can be simply separated in to the two categories: mobile and desktop. Mobile experiences such as the Google Cardboard and Samsung's Gear VR target the audience which already own a compatible mobile device thus eliminating the cost of hardware found in higher end platforms. Through the use of the phone's built in gyroscope and accelerometer, crude head tracking can be achieved to emulate a virtual world.

High end virtual reality platforms target enthusiasts and early adopters of cutting edge technology due to its premium price and high computer hardware requirements in order to run it. Currently there are two virtual reality headsets that are seen as the devices that give highest immersion and these are Facebook's Oculus Rift and HTC's Vive. These will be discussed later in this chapter.

2.1.1 Mobile VR

On the market right now there are two different Mobile Virtual Reality hardware. There is the Samsung Gear VR and the Google Cardboard.

Google Cardboard

The Google Cardboard is the cheapest Virtual Reality headset out on the market right now, but it does come with the least features out of them. The cardboard viewer is a stereoscope made out of cardboard. It contains two 40mm focal lenses that are designed to give a distortion when looking through them, which is counter-acted by the distortion from the application[4].

To use the Google cardboard you would need to install the cardboard application on your compatible phone and then place your phone inside of the Google Cardboard. Once the phone is inside the Cardboard it uses the phone's inertial measurement unit to track head movement. This does have limitations however as the Google Cardboard does not track displacement if the user was to walk in any direction.

The Google Cardboard still uses the technology of stereoscopic images, as can be seen in 2.2. Although it now does it with moving images, which creates a more immersive experience.

Figure 2.2: Image showing the Cardboard demo application

The Google Cardboard was not chosen to the virtual reality device for this project as there are many drawbacks to it, and as such it does fully demonstrate all the features present in modern technology for virtual reality. The drawbacks to the Google Cardboard are:

- No displacement tracking, making it less immersive than the other options
- Only one input method, a button on the cardboard which acts as a screen press.

Samsung Gear VR

The other mobile Virtual Reality headset on the market is the Samsung Gear VR. The Samsung Gear VR is slightly more expensive than the Google Cardboard, and as expected with the price increase, it comes with more features compared to the Google Cardboard.

The Samsung Gear VR uses the same technology as the Google Cardboard in the sense that it uses stereoscopic imaging to create the illusion of depth. This is done in the same way for both VR devices, by inserting a compatible phone into the phone holder in the headset, and then showing the stereoscopic images on the phone screen. As seen in 2.3 the Samsung gear VR uses the same stereoscopic technology as the Cardboard uses, as seen in 2.2.

The Samsung Gear VR also uses an inertial measurement unit to detect head movement, similar to the Google Cardboard. The Samsung Gear VR uses an inertial measurement unit contained in the headset, rather than using the attached phone's inertial measurement unit. The inertial measurement unit contained in the headset is more accurate, has lower latency, and is better calibrated than standard phone inertial measurement units, as it uses the same I.M.U. as the Oculus Rift. This I.M.U. is more accurate as it has a higher sample rate than internal phone I.M.U.s and therefore gives it more values to use, so that it can more accurately detect erroneous values.

Figure 2.3: Image showing the Samsung Gear VR menu

The Samsung Gear VR has a few extra features compared to the Google Cardboard, for example when a phone is placed inside the Galaxy Gear VR it needs to be connected by a micro-usb connection, which allows the headset to have more input methods to the phone, as well as giving access to the headset's I.M.U. The extra input methods that the Gear VR has access to are:

- A home button, which works the same as the home button on Android phones.
- A back button, which works the same as the back button on Android phones.
- A touch pad, which works by swiping to move across menus, and tapping clicks the highlighted item in a menu.

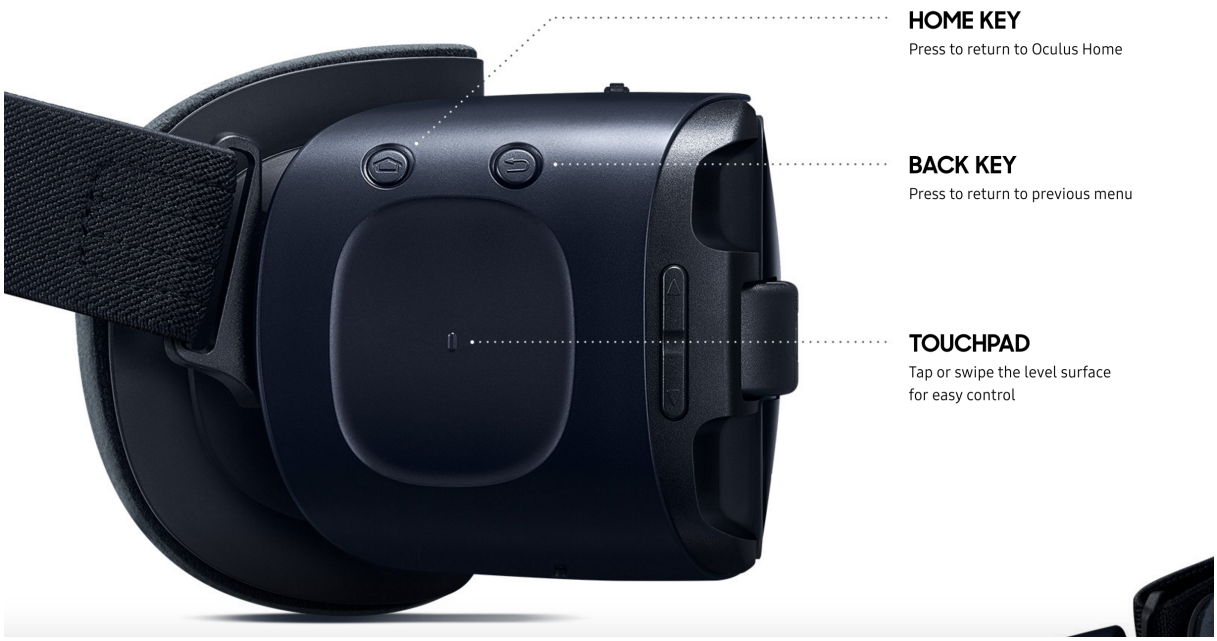

Figure 2.4: Image showing the hardware controls on the Samsung Gear VR

The Samsung Gear VR will not be used for this project as again it has several drawbacks, which are:

- It only tracks rotational movement, not displacement, which makes it less immersive than the other options
- The Samsung Gear VR has very primitive control, which are only the buttons and touchpad on the side of the headset

2.1.2 Oculus Rift

The Oculus Rift was the first of the two to be released and is inferior in terms of the level of immersion that can be achieved, as currently Oculus only supports interfacing with the virtual world through a third party traditional controller that simply uses buttons and joysticks. The Oculus Rift tracks by using the single camera to pick up infrared light that is emitted by points on the headset. These can be used to track the headset as they blink in a specific pattern, which the sensor knows, and it then uses that to determine the position the headset is in.

2.1.3 HTC Vive

The HTC Vive works using two base-stations. These emit lasers in an alternating pattern, between vertical and horizontal. If these lasers hit a sensor on the headset or controllers, they emit a pulse. By tracking the timings of the laser sweeps and the emitted pulses, the tracking system can use trigonometry to find the position of the location of every sensor on the devices [11]. The HTC Vive has two settings, either a sitting or a standing mode. in the sitting mode, it works similar to the oculus Rift, in that a console controller is used to control the game, where as in the standing mode, it uses it's own controllers, which are tracked by the base stations and provide a more immersive experience as you can interact with objects in the game by using these controllers to pick things up by moving your hands to where the object is in game.

2.2 Development Environments

To develop on the HTC Vive there are two options which are currently supported and these are the Unity engine and the Unreal 4 engine. These game engines has native support for SteamVR which is the platform developed by Valve that powers the Vive. Both are free to be installed and just requires a simple registration to their respective websites.

Unity supports the C# programming language for development whereas Unreal Engine uses C++ on its current version of the software. Another aspect where it differs is Unity is mainly used to develop games for mobile devices such 2D platformer games. Unreal Engine is mainly used for desktop, AAA games which makes it more suitable for virtual reality with games running on Unreal generally looking better.

Unreal Engine includes many features built in such as particle effects simulation, terrain, lighting and shading [1]. With SteamVR being natively supported, simple virtual reality features such as gesture recognition and teleportation movement mechanic can be easily implemented in to a game.

For this project Unreal Engine will be used as the development environment. This is because the group is more comfortable with using C++ so development will be quicker with Unreal rather than trying to use a language with little familiarity.

2.2.1 Unreal Engine 4

The Unreal Engine offers many features for people with little to no programming experience, these will not be used however, as they do not demonstrate any Computer Science expertise, one of the many features that will not be used is the drag and drop interface that is packaged in Unreal Engine to develop simple software quickly, along with the basic templates for various types of games. These are templates for popular genres, for example First Person Shooter and Sidescroller. For this project the group will implement their own features and backend for the genre that is picked, in order to demonstrate their ability to code for the HTC Vive. A select list of the features that Unreal Engine implements already are as follows:

Unreal Engine 4 Features

- Particle Effects Simulation (Visual Effects)
- Procedural Foliage
- Landscaping/Terrain
- Lighting
  - Directional
  - Point
  - Spot
  - Sky
  - Shadow Casting
- Shading
- Post Process Effects

  - Bloom
  - Ambient Occlusion
  - Colour Grading
  - Depth of Field
  - Lens Flares
- Material Effects
- Fog Effects
- View Distance Culling
- Distance Dependent Level of Detail Models
- Physics Simulation
- Level Streaming (Ability load and unload map files into memory and toggle their visibility)
- Basic Templates
  - First-Person
  - Third Person
  - Side Scroller
  - Vehicle
- Artificial Intelligence System
- Audio System
- DirectX 11 & 12 Features
  - Full-scene HDR reflections
  - Per Scene Dynamic Lights
  - Physically Based Shading

Unreal Engine's full list of features can be found on their documentation site [2]

Chapter 3

Requirements

3.1 Client Requirements

The client has asked us to produce this technology demonstration in order to have a piece of software for the HTC Vive to demonstrate to students and has been developed by a University of Leeds student. The requirements that were given by the client were to appeal to the target audiences (who will be discussed in the next subsection), and to fully utilise the functionality of the HTC Vive in order to properly demonstrate its capabilities.

3.1.1 Target Audience

The demo would be targeted towards potential students looking to apply to the University of Leeds, as it will be shown to the students on Open days in order to gain interest from them. It would also have to appeal to people and companies in the gaming industry. As the more interest the School of Computing can gain from them, the more potential projects they may have for the school. With these target audiences in mind the demo should be technical to impress the gaming industry, while balancing it with being interesting for the potential students.

3.2 Feasibility Assessment

3.2.1 Feasibility

To support the development of the project, a reserved space where the HTC Vive can be permanently set up for the duration of the project is required. This reserved space would ideally be a room that meets the space requirements stated above for room-scale experiences, so that all the capabilities of the Virtual Reality hardware can be used. A computer which meets the hardware requirements is also needed in order to run the HTC Vive software, This computer must be running Windows since HTC Vive currently only supports this operating system [5]. Also, a copy of Unreal Engine 4 game engine must be installed which is free to be downloaded. Unreal Engine 4 is chosen over Unity since the members of the group are more familiar with developing in C++ which Unreal Engine supports rather than C# which Unity supports, although both of these game engines provide native support for virtual reality developments.

A possible solution for meeting the hardware requirements is to use a personal machine. A laptop is available with the specifications below which just meets the requirements for the graphical power. Development can be done on our own personal Windows machines and can be tested with the Vive using the laptop in the reserved room. The Vive is not required for conducting simple tests, but it is needed for identifying issues such as scaling and user input with motion controllers.

3.2.2 Technical Specifications

Hardware Requirements

- Graphics card: NVIDIA GeForce GTX 970 /Radeon R9 280 equivalent or greater

- Processor: Intel Core i5-4590 equivalent or greater
- RAM: 4GB+
- Video Ports: HDMI 1.4, DisplayPort 1.2 or newer
- 1x USB 2.0 port
- Room-Scale Space: 2 meters by 1.5 meters

These requirements can be found on the official HTC Vive page on the Steam Store [10]

Chapter 4

Project Management

4.1 Methodology

The methodology that will be used for this project is the Agile methodlogy. This methodlogy will be used for this project as following the agile methodlogy gives you a working product earlier in the building phase. Agile also requires a weekly standup to be held, this will keep the project on schedule, as during this standup meeting the week's progress on the project will be checked and evaluated.

4.2 Schedule

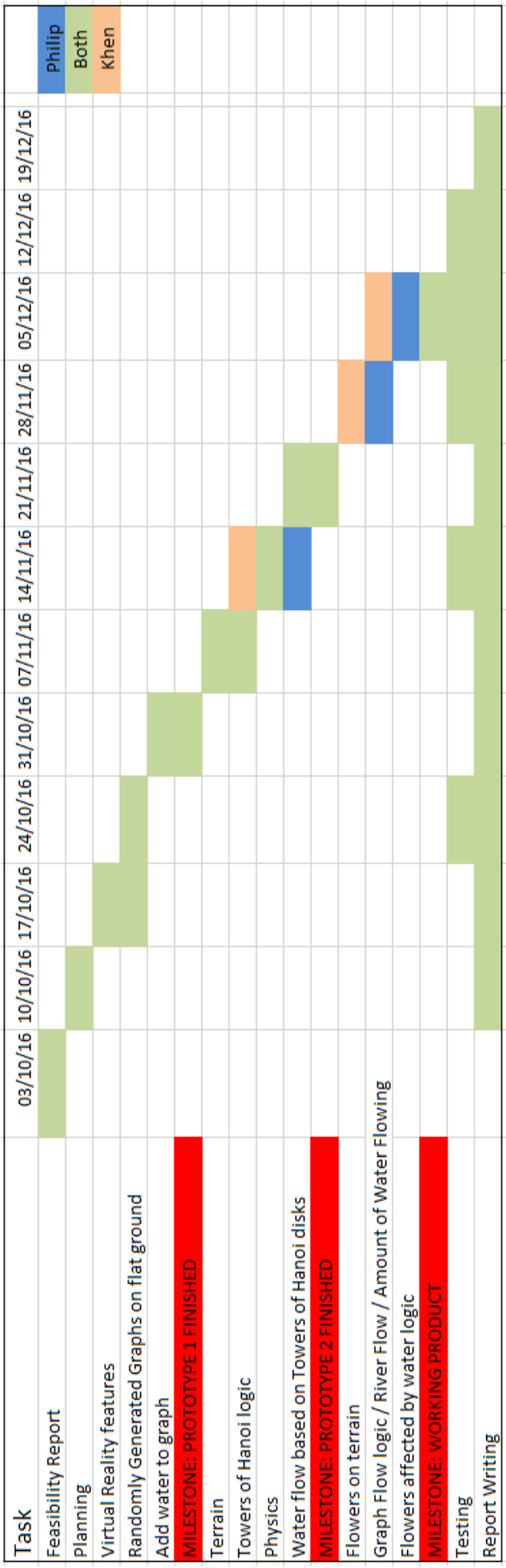

Figure 4.1: Gantt Chart showing the work that will be done each week.

Writing Milestone	Date For
Layout Done	14/11/2016
Chapter 1 - Introduction	21/11/2016
Chapter 2 - Background	28/11/2016
Chapter 3 - Requirements	28/11/2016
Chapter 4 - Project Management	05/12/2016
Chapter 5 - Planning and Design	05/12/2016
Chapter 6 - Implementation	12/12/2016
Chapter 7 - Testing and Evaluation	12/12/2016
Chapter 8 - Conclusion	12/12/2016
First Draft Complete	12/12/2016
Report Finished	23/12/2016

4.3 Version Control

The Version control for this project will be done on github. Version control software will be used for the project in order to accurately document any and all changes that will be made to the code.

4.4 Risk Assessment

A risk involved with this project is the availability of the HTC Vive headset as well as its peripherals. Since the tech demo will be making use of the Vive as the virtual reality hardware, it is essential that a Vive is at hand to be used for testing the demo. The school currently owns 2 Vive packages and since the school is currently only using them for demoing purposes of off the shelf demos and not for development, a Vive should be available for the development in this project. If there is no access to a HTC Vive headset, a contingency plan has been made, which is outlined in 4.4.1.

Another risk that may occur is that the Vive has certain requirements for the space it needs to use for a room-scale experience. The Vive takes some time to set up since it has to recalibrate to the environment every time it is set up again. This means a dedicated room where the ViveâĂŹs base stations can be left set up is ideal. If a big enough room is not available, Vive has an option to use a standing/seated experience instead. This will means that user movement cannot be tested and movement in the virtual world will have to be restricted to ViveâĂŹs teleportation mechanic. The user looking around the world will still be supported with this version.

One other factor that needs to be taken in to consideration is the availability of computer hardware that is powerful enough to run and develop on the HTC Vive. Since the HTC Vive has high minimum requirements to run it, finding a computer that is sufficient and can be used when needed can become an issue.

4.4.1 Contingency Plan

In the case that there is no access to a Vive Headset for this project, due to the risks mentioned before, a contingency plan has been made. The plan would be to make a desktop application using the same premise, rather than being able to use the Vive for the game. This desktop application would have the same features as the Vive version, although without the VR features, i.e. looking around using the headset and interacting with objects using the controllers.

The plan for dealing with if there is no room available is that a seated experience could be developed without need a room set-up. This would just involve setting up one camera above the computer and it would only track head movement. A standard game controller would have to be used for this set-up.

For the risk that there is no high-end computer available, a high-end laptop that can run the software has already been acquired and will be used if there is no other computer available.

Chapter 5

Planning and Design

5.1 Game Design Process

The game design process started by using the HTC Vive to research what kind of games were already on the market for the HTC Vive. The games that were SteamVR Demo[9], The Lab[8], Space Pirate Trainer[6], and Elite Dangerous[3]. This research was done in order to give a feel for the HTC Vive and it's capabilities, this would give a clearer understanding to which kind of games are more suited to the Virtual Reality platform.

The next step in the game design process was to brainstorm ideas of what kind of game would be made for the project. The ideas would be split into four categories, these four categories are the theme of the game, the features of the game, the genre of the game, and the benefits of the game as seen in the whitebaord Figure 5.1 and the table bersion in Figure 5.2.

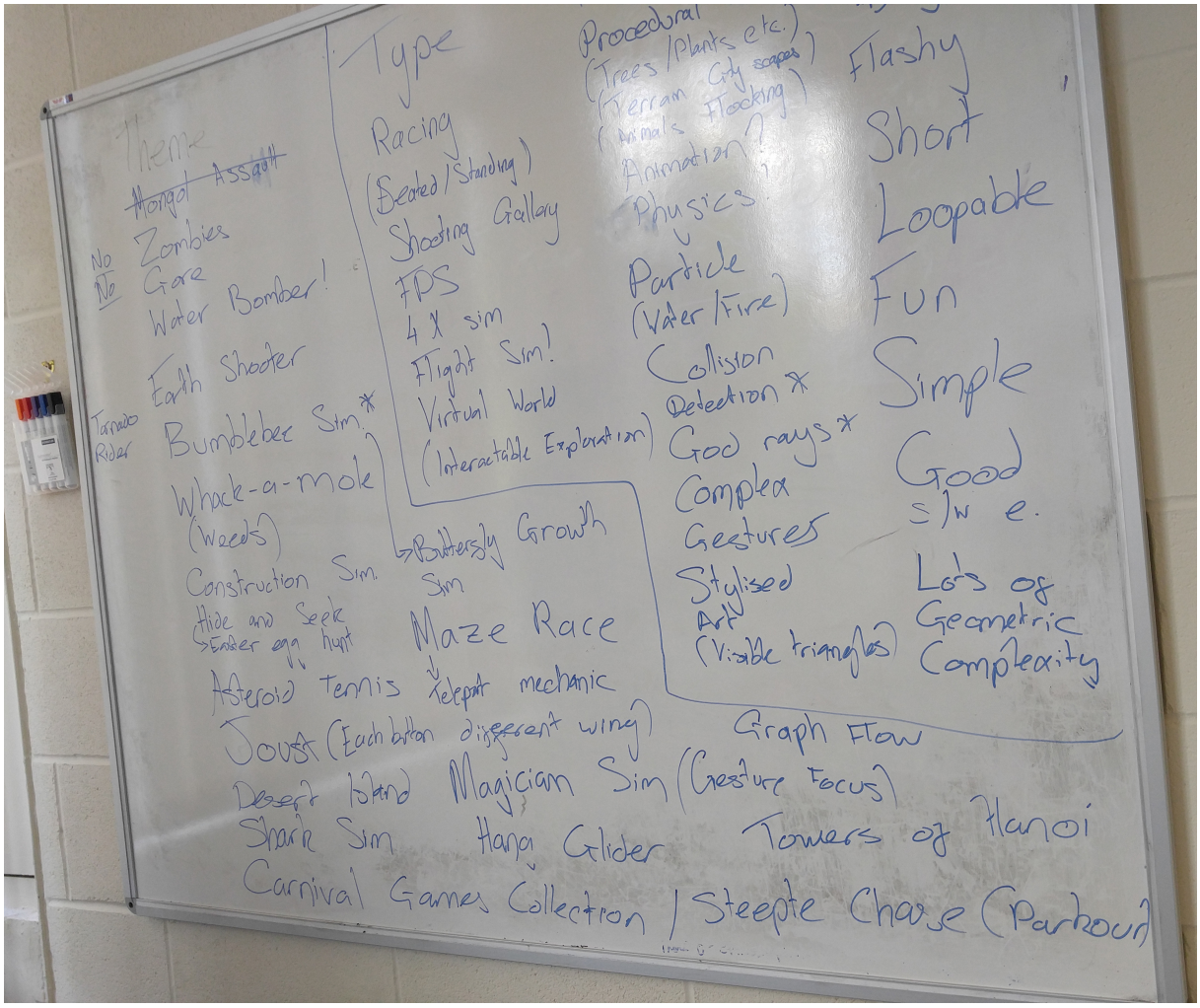

Figure 5.1: Image showing the whiteboard brainstorm for demo ideas.

Theme	Type	Features	Goals
NO ZOMBIES	Racing	Procedural Generation (Plants / Terrain / City Scapes / Animals)	Techincal
NO GORE	Seated	Animation	Flashy
Water Bomber	Standing	Physics	Short
Earth Shooter (i.e. Worms / Tanks)	Shooting Gallery	Particles (Water / Fire)	Loopable
Bumblebee Sim	FPS	Collision Detection	Fun
Butterfly Growth Sim	4 X Sim	God Rays	Simple
Whack-a-mole (Weeds)	Flight Sim	Complex Gestures	Good Software Engineering
Construction Sim	Virtual World (Interactable Exploration)	Stylised Art (i.e. Visible Triangles)	Lots of Geometric Complexity
Hide and seek / Easter Egg Hunt			
Asteroid Tennis			
Joust (Each button controls a seperate wing)			
Desert island			
Shark Sim			
Carnival Games Collection			
Maze Race (Could use teleport mechanic)			
Magician Simulator (Heavy gesture controls)			
Hang Glider			
Graph Flow			
Towers of Hanoi			
Tornado Rider			

Figure 5.2: Table showing the brainstorm for demo ideas.

The brainstorm was then categorised in a separate table, combining the ideas that have relevance together, this table can be seen in 5.3.

Theme	Features	Goals Met	Goals Not Met
Water Bomber	Procedural Terrain	Loopable	Short
	Particles	Simple	
	Basic Collision Detection		
	Physics		
Earth Shooter (i.e. Worms / Tanks)	Procedural Generation (Terrain / Foilage)	Loopable	Short
	Animation	Simple	
	Physics		
	Particles		
Bumblebee Sim	Collision Detection	Loopable	
	Procedural Generation	Simple	
	God Rays	Short	
Butterfly Growth Sim	Collision Detection		
	Procedural Generation		
	God Rays		
Whack-a-mole (Weeds)	Collision Detection	Loopable	
		Simple	
		Short	
Construction Sim	Complex Gestures	Simple	Loopable
	Physics		Short
	Collision Detection		
Hide and seek / Easter Egg Hunt	Gestures	Loopable	
	Procedural Generation	Simple	
		Short	
Asteroid Tennis	Physics	Loopable	
		Simple	
		Short	
Joust (Each button controls a seperate wing)	Animation	Loopable	
	Physics	Simple	
	Complex Gestures	Short	
Desert island	Procedural Generation		
Carnival Games Collection	Physics	Loopable	
	Complex Gestures	Simple	
		Short	
Maze Race (Could use teleport mechanic)	Procedural Generation	Loopable	
		Simple	
		Short	
Magician Simulator (Heavy gesture controls)	Complex Gestures	Loopable	
		Simple	
		Short	
Hang Glider	Procedural Generation	Loopable	
	Complex Gestures	Simple	
		Short	
Graph Flow	Physics	Loopable	
	Animation	Simple	
	Particles	Short	
	Procedural Generation		
Towers of Hanoi	Physics	Loopable	
	Procedural Generation (Terrain / Foilage)	Simple	
		Short	

Figure 5.3: Table showing brainstorm ideas categorised and coloured depending on how probable it would be used.

Then using the updated table of ideas, the ideas were narrowed down until one that could feasibly done was chosen out of all the ideas on the white-board. This idea was to combine the Towers of Hanoi and Graph Flow in a puzzle game. These two ideas were combined in order to make a technology demonstration that is very specific to the School of Computing in the University of Leeds, as these are both topics of study during the Computer Science course in the university of Leeds.

A goal for the game then had to be designed. The goal of the game was decided to be having the right amount of flow run down to set nodes, where flowers would be. If these flowers have too much water flow they would look dead, and if they had too little water flow they would also look dead. They would only look alive if the water flow was the correct amount. The game is won when all the flowers are "alive" at the same time.

This idea would implement many non-trivial features in the Unreal Engine, such as:

- Randomly generating a graph
- Generating a terrain based on the graph
- Towers of Hanoi logic for flow control
- Having plants being affected by the water flow
- Changing the appearance of the water, based on the flow of that edge of the graph

5.2 Virtual Reality Features

The Virtual reality Features that will be used are:

- Using the Head Mounted Display in order to look around the virtual world.
- Using the controllers to pick up the Towers of Hanoi disks, and place them down.
- Using the controllers to select a teleport location to move.

5.3 Graph Flow

Graph Flow will be used in this technical demonstration as the main objective of the game. The objective of the game will be to get the flow to match the goal flow. This will be done by blocking off certain rivers to redirect flow from one river into another. The goal flow will be calculated in development, and each level that is created will also have a goal state completed.

5.4 Towers of Hanoi

Towers of Hanoi will be used in the demo to block off the graph flow from nodes to other nodes. This will be done by having rods in the river at each split in the graph. These will be placed at the output of each split in the river. The Towers of Hanoi disks will then be placed on top of these rods if the flow needs to be blocked. The Towers of Hanoi disks will only be allowed to be placed in the opposite order compared to the way the normally are. This would mean that a larger disk can only be placed on top of a smaller disk. This order had to be inverted to the standard as rivers are narrower at the bottom, compared to the top of the river ditch.

5.5 Plant Survival

The Plant Survival will be used as a measure for the player to determine how close he is to solving the puzzle. This will be the indicator to the player if the flow at that node matches the pre-determined flow for the level

Chapter 6

Implementation

6.1 Development Environment

The development of this project was done in various environments, as there was many different aspects to this project.

6.1.1 Vive Hardware

For this project, the Vive Hardware was set up in a dedicated room (The Virtual Reality Lab in the School of Computing). This allowed development to be done on the HTC Vive without having to set up the sensors and calibrate the hardware every time that development had to be done, which maximised the amount of development time that was available.

Figure 6.1: Room used for testing the application

This room was chosen as it was unused and met the space requirements for the HTC Vive. Approximate room space available resulted in 2.9m x 2.9m which is more than sufficient.

6.1.2 Game Engine

The chosen game engine used for development as stated in chapter 2 is Unreal Engine. The latest version as of the time of development was used which was Unreal Engine 4.13.2. This decision was made to make sure bugs that may have appeared in previous versions were fixed and also for new features to be taken advantage. Since virtual reality is bleeding edge technology, new features and ways to use VR in development are being implemented rapidly. When development started an older version was being used and new updates were being rolled out, the group decided to update the developement environment when a new update was released. It was made sure that everyone in the group was using the same version of Unreal to avoid backward compatibility issues.

6.1.3 Visual Studio

When developing in Unreal Engine, the editor used was Visual Studio. Unreal Engine was designed to integrate smoothly with Visual Studio so code changes can be built in Visual Studio and then can be quickly seen in the Unreal Editor. It also provides a debugging tool to help solve code problems and bugs due to the fact that it has knowledge of the Unreal API. Visual Studio's Intellisense feature has the ability to list members, methods and parameter information when coding. This helped with learning the API, making it easier to detect syntax errors before compiling. The version used was Visual Studio 2015 since Unreal Engine verisions 4.10 or later must use this version and not the older 2013 version.

6.1.4 Windows

The operating system used was Windows 10 since the SteamVR software that powers the HTC Vive is only compatible on Windows operating system as of the time of development. Windows 10 was the version that was the latest version and the personal systems the group members owned that were used to develop was already running Windows 10 which made it more convenient to use.

6.1.5 Out of Engine Development

The out of engine development was done in C++, using Sublime Text as an editor, this decision was made out of familiarity with the software. Some C++11 functions and types were used in the development of the graphs, rivers and terrain. The generation for these was done without the use of the Unreal Engine as it outputs files that are then read in with the Unreal code to help generate the game terrain, objects and logic.

6.2 Random Generation of Graphs, Rivers and Terrain

6.2.1 Graph Generation Original Method

Point Insertion

The original idea to do graph generation was to start with a square, using each corner as a node. These nodes would then be connected as shown in 6.2.

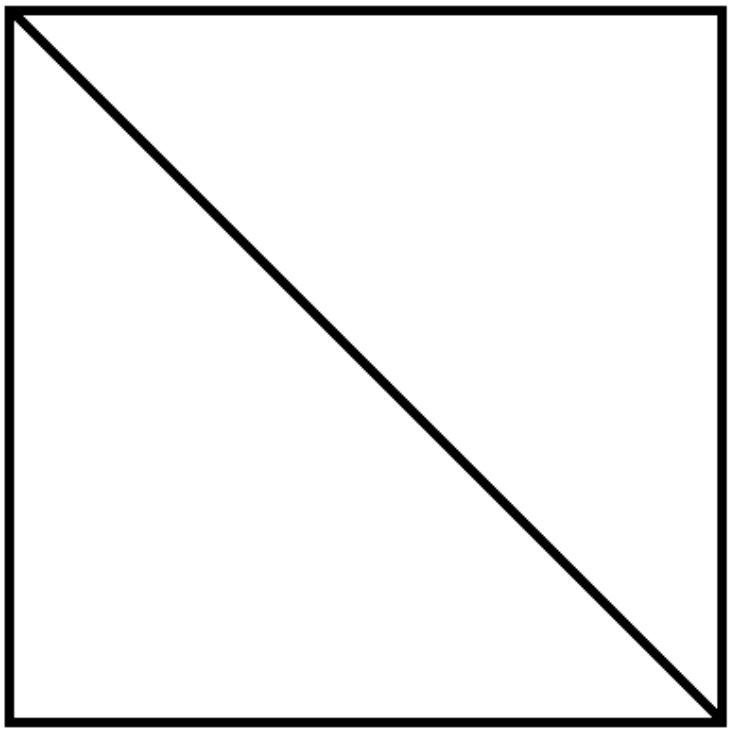

Figure 6.2: Original Connections

Points would then be inserted into this square, using a random x and y value. The triangle that this node is in would then be found using the cross product. This was done by checking the cross product of the point and the triangle, for each triangle that is in the graph. The point would then be connected to the corners of this triangle.

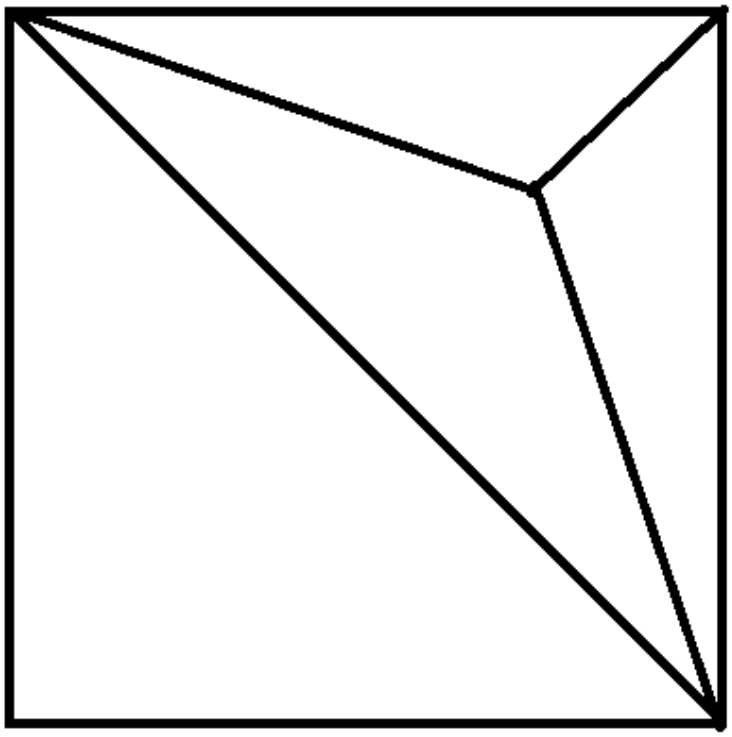

Figure 6.3: Example of running the point insertion algorithm for one iteration.

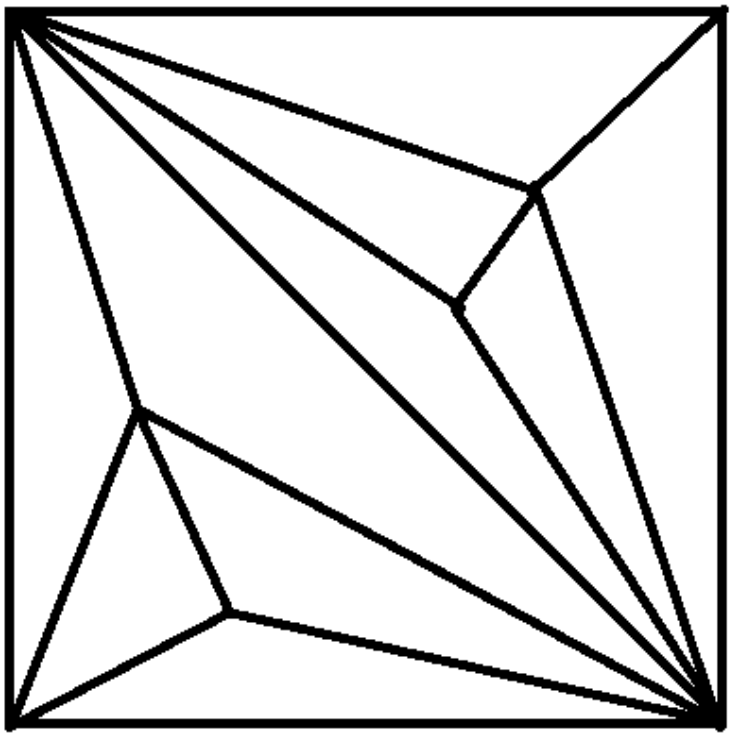

Figure 6.4: Example of running the point insertion algorithm for four iterations

This approach guaranteed a connected graph to begin with. This approach did not work however as if a node on the bottom-left side of the graph needed to be connected to a node on the top-right side of the graph, the the connection would have to be made through either the top-left node or the bottom-right node, as there was no other way to pass through to the other side of the graph. The approach was then modified slightly to start with an extra node node in the middle of the square, allowing another way to pass through the other side of the graph, this is seen in 6.5.

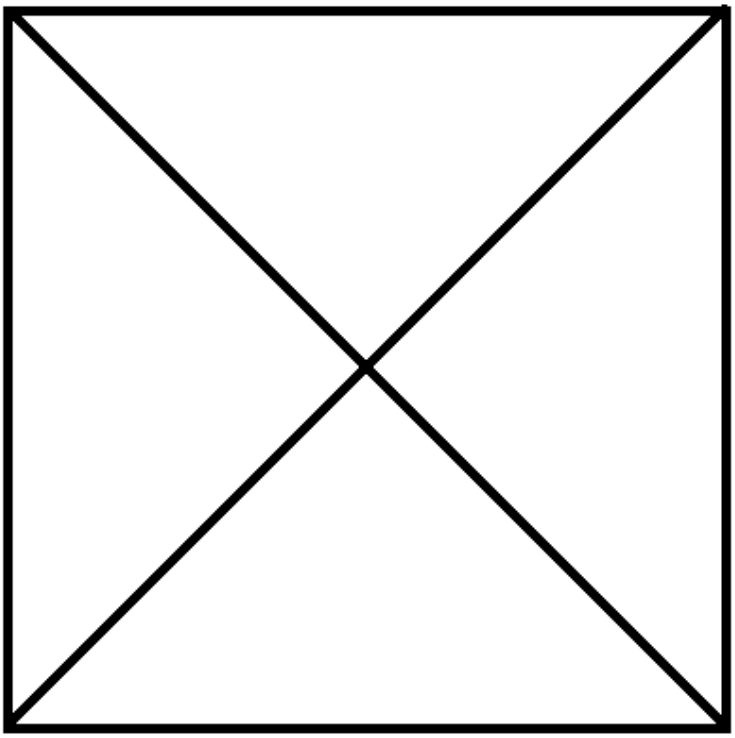

Figure 6.5: Image showing the second method's starting connections

This approach also did not work, as the addition of the extra node did not provide enough of relief for the connections between the two sides of the graph, and the connections would occasionally still go through the top-left or bottom-right node. The next approach was adding several nodes along the diagonal and connecting them to the corners, similarly to how the middle node was added. The third approach can be seen in 6.6.

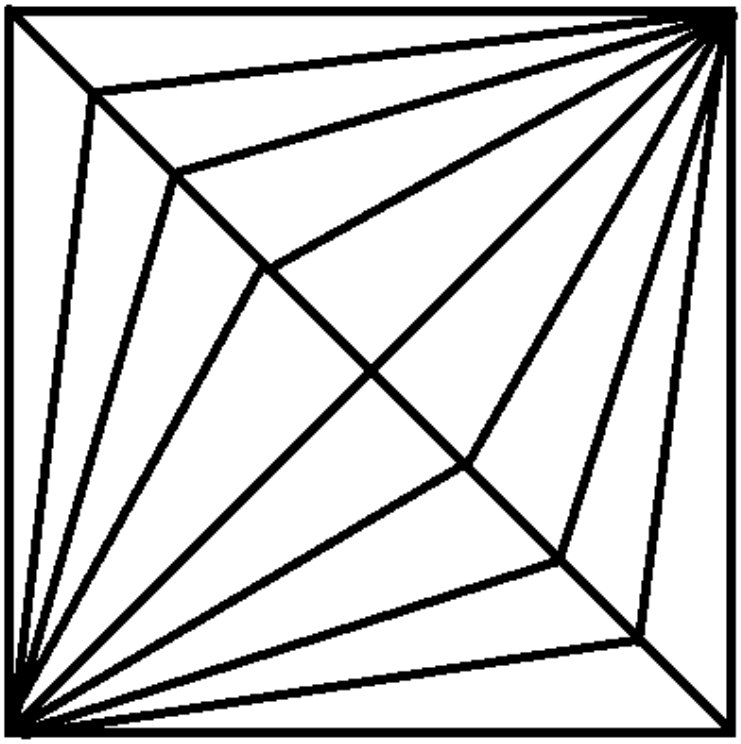

Figure 6.6: Image showing the third method's starting connections

This approach also did not work as when the shortest paths between nodes were found, the paths tended to favour the path from the top-left to the top-right. This would make the graph just be a straight line, with a few edges going to the nodes that were used in the river graph, an example of this can be seen in 6.7.

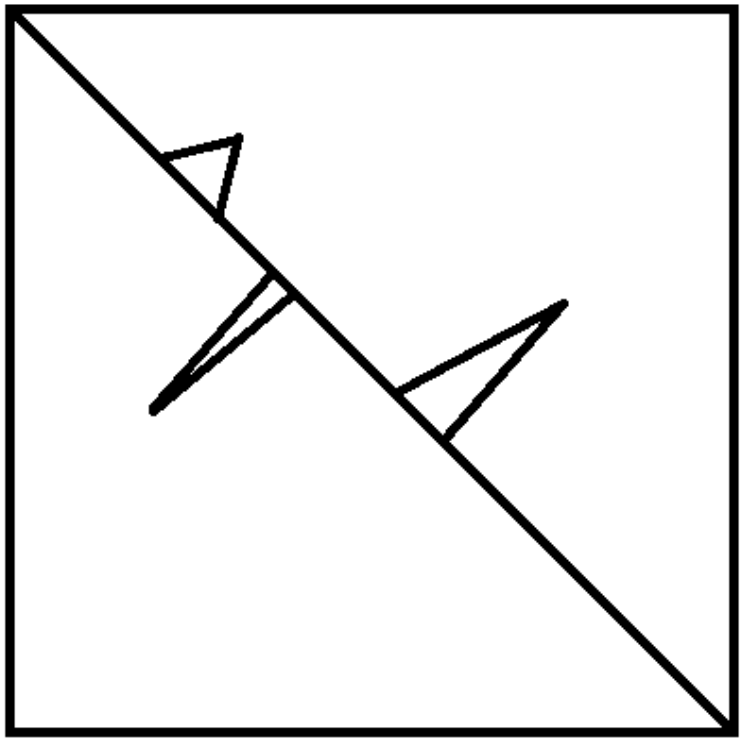

Figure 6.7: Image showing an example of the third method's output

The next method that was developed was an entire overhaul of the system, and this will be outlined in 6.2.2

Connections and Weight Matrix

The connections were found by looping through the array that stored the triangles and finding the edges of the triangles. These edges were stored in a connection matrix, to be used when making the weight matrix.

The weight matrix was made using Manhattan distance between the two connected points as the heuristic. The Manhattan distance is the x distance between the two points plus the distance in the y direction. The Manhattan distance was calculated using the following formula:

$$Manhattan = (point1.x - point2.x) + (point1.y - point2.y)$$

The Manhattan distance then has to be checked to see whether it is positive or not. If it is positive the result is put in element of the matrix representing the first point to the second point, if the result is negative the result is instead placed in the element representing the second point to the first point. This is repeated for each connection in the connection matrix produced before.

Generating Rivers

The river generation starts off with picking the start and end nodes for the rivers, this is done by randomly selecting a set amount of nodes, determined by a variable set in code. The top-left node and the bottom-right node are also selected, as they will be the input and output nodes for the graph. These nodes are then looped through one by one, and connected to two other nodes, that are closer to the bottom-right. Nodes are determined to be closer to the bottom-right by adding the x and y coordinate of each node together, and then comparing them. This value will be larger the closer to the bottom-right the node is.
After which nodes are connected has been determined, dijkstra's algorithm is used to find the shortest path between each pair of connected nodes.
Using the shortest paths generated by dijkstra's, a new connection matrix is made. This connection matrix is made by looping through each path made by dijkstra's algorithm and looking at each pair of nodes. These pairs would be the consecutive nodes in each path. After this connection matrix has been made it is time to map out the graph, so that it can be transferred to the terrain. Mapping out the graph is done by having a matrix of the same size as the terrain will be, this matrix is initialised to have all ones, as the lines will be modify the elements to be 0, as they will be ditches. Then by looping through the connection matrix made before, the connections will be drawn onto the matrix. The connections are drawn onto the matrix by first checking if the element in the connection matrix is 0 or 1, 1 being that there is a connection between the two nodes. If there is a connection then the line between these two points should be drawn onto the height map matrix. This line is calculated by using a modified Bresenham's Line Algorithm, the algorithm is modified so that it can be used in all eight octants of a graph, rather than just the octant that the line goes down and to the right. This modified algorithm is explained in more detail in the section below.
After the lines have been drawn, it is time to expand them on the height map, this has to be done so that the connections look more like rivers in the demo. This widening is done by taking a copy of the height map and using the copy to modify the original. The copy of the height map is then looped through, checking each element of the height map. If the original height map is equal to zero, meaning that a line has been drawn in the element, then the algorithm will try to change the value of the pixels

in set distance away from this pixel in both the x and y direction, this distance is set as a global variable. The algorithm changes the values of the elements of the copy height map within the range depending on how far away the elements are. Within a third of the distance away the value is set to zero, within two-thirds of the distance then the value is set to 0.33 and just within the distance it sets the element to 0.66, this is done as in the game the Towers of Hanoi discs will need to fit inside the rivers, and have their own platform to rest against, and as there are three discs, the distance is split into thirds. The original height map is then replaced by the copy of the height map.
The generated height map with the rivers on it is then sent to be merged with the randomly generated terrain.

Bresenham's Line Algorithm

Bresenham's Line Algorithm is a rasterisation algorithm, it works using error checking for the y coordinate. The original Bresenham's Line algorithm works by finding the absolute value of the gradient, then it loops through each point in the x direction of the line segment. At each iteration through the line it adds the gradient to an error value, when this error value ticks over the value 1, it will increment the y value of the coordinate for the rest of the iterations. At each iteration of the algorithm, it will plot the point of the current x and y coordinate onto the height map.
The modified Bresenham's Line Algorithm that is used in this project does the same thing, but it lets the algorithm work for lines that are not sloping down and to the right. The modified version does this by first checking the gradient and if the gradient is over 1 or under -1, then the algorithm will swap the x for the y values and the y values for the x values, so that the gradient is less than 1 and greater than -1. The algorithm will then check if the line is sloping to the left or right, the check is comparing the x value of the first point to the x value of the second point and seeing which is bigger. If the line is sloping to the left, the algorithm will swap the points, so that the first point becomes the second point, and the second point becomes the first point. The last step the algorithm takes to ensure that it will work, is that it will check whether the line is sloping upwards, or downwards. If the line is sloping upwards, it will set the change in y to be -1, so that it subtracts from the y value, rather than adding it.

6.2.2 Graph Generation

As the original method did not work, parts of the original algorithm were reworked, in order to give a better result for the river generation.

Start Point Generation

The reworked method to generate the starting graph was to start with an empty matrix of nodes, of a size determined by a global variable. These nodes are then all connected using 8-connectedness. This means each node is connected to the eight nodes around it. This makes a graph that looks similar to 6.8.

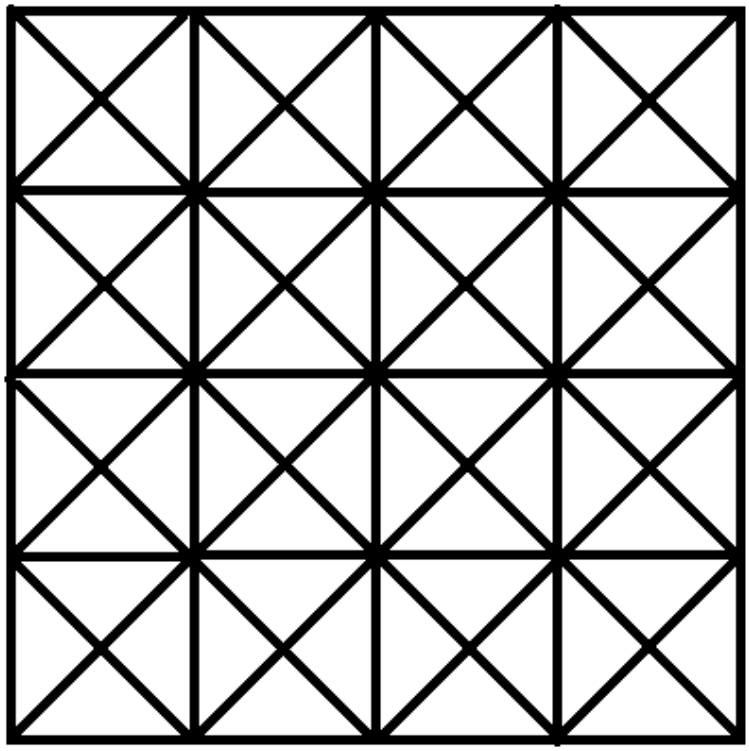

Figure 6.8: Image showing an example of the new method of connection the nodes, where the starting matrix is 5x5.

This produces a connected graph which contains several edges for the connections between nodes to use.

Connections and Weight Matrix

The starting set of nodes is found in the same way as before, which is selecting the top-left and the bottom-right point as starting nodes, and then randomly selecting others until the amount of nodes selected is the same as the variable stating how many nodes should be selected, and the way that the connections between nodes were found has already been described. The weight matrix is also determined in the same way as before, as described in 6.2.1.

Generating Rivers

To generate the rivers the algorithm must first decide which nodes should have rivers running between them. It does this by first finding the distance from the top-left to each of the selected nodes. It then loops through the selected nodes and tries to find two connections where the flow is going downwards. It makes sure the flow is going towards the bottom-right by checking the distances, and if the distance is larger on the node it is trying to connect to, then the river is flowing to the bottom-right. There is a for-loop that will loop through each node, in the order of smallest distance from top-left to largest distance from top-left, looping in this order gives the benefit that the closest nodes to the current will always be selected to connect.
The next step in generating the rivers is to find the shortest path between the connected nodes. This is still done with Djikstra's algorithm, see the explanation in 6.2.1.
The coordinates are then updated so that the coordinates align with the size of the grid, instead of just being consecutive numbers. The coordinate updating is done by first getting the scale factor for the points. The scale factor is the size of terrain divide by the size of the starting matrix, rounded down. The coordinates are then looped over and each x and y coordinate are multiplied by the same scale

factor, as the terrain is always square. The last point (the bottom-left point) is then set separately, as it should be set to the bottom-left of the terrain.
After the coordinates have been updated it is time to generate the connections matrix, this is done the same as before, by looping over the shortest paths and setting each element in the matrix, that represents the consecutive pairs in the shortest path, to be connected. While the generation code is doing this, it also is determining which nodes should be the "new" nodes. These "new" nodes are where the intersections are in the graph, as the previous nodes are not always guaranteed to be at an intersection. These "new" nodes are used later on in the program in order to decide where to place the rods in the game.

The program determines which nodes by checking which nodes have been used two times or more. It does this during the for loop when it generates the connection matrix, whenever a element in the connection matrix is set to be connected, then it loops over the array that stores all the connections that have already been used, and checks if the first point of the used connection is the same as the first point in the connection that has just been stored in the connection matrix, it also checks to make sure the second points are different. If the two first points are the same and the two second points are different, then an intersection has been found. The algorithm will then check the array of already found "new" nodes, to make sure that the node doesn't already exist in the array. If it does not exist in the array, it is added to the array, the used connection is then added the array storing all the used connections.
After the previous step comes removing edges from the graph that either act as either a sink for the graph, i.e. there are no connections after it, or it acts as a source for the graph, i.e. there are no connections leading to it, as the graph should only have one source, the top-left, and one sink, the bottom-right. The first step in this algorithm is to determine how many times each node has been used as a starting node (being used as the first point in a connection), and how many times it's been used as an end node (being used as the last point in a connection). This is done by looping over the used connections array from the previous step, from each element of this array the start node and end node is extracted. The corresponding nodes then have their respective values incremented in an array for both start nodes and end nodes.

One of the two cases that are being removed is when a node is not used as an end node, but it is used a start node one or more times. In this case the node would be a source node. This node is then placed in an array of nodes that are removed the graph, and should not be included in it. Now the algorithm needs to look at the nodes that the removed node was connected too, and see if they should be removed. This is done by implementing a stack. The nodes that are connected to the removed node are placed in the stack. This is implemented by having two while loops. One of the while loops is checking to see if the stack is empty, the other is checking if the end of the path has been reached, this being the inner while loop. The end is found when the algorithm can not remove any more nodes from that path. Inside the while loops there is a for loop, looping over the used connections array. For each iteration of this for loop, the start and end node are extracted. The start node is then compared against the node that is being looked at by the algorithm, if they are the same it then checks if there is more than one end node and if the node does not exist in the removed array. If these are both true then the algorithm adds the node to the removed array. The algorithm then needs to find the nodes that are connected to the node that was just removed. The number of connections is found by looping through the connections array and extracting the start node of each element, and incrementing a counter every time the start node matches the removed node. If only one connection is found then the inner while loop carries on, but the current node is now the second point in the connection that was being looked at when the node was removed. If there is more than one connection then every connection found after the first one will have the end node extracted and added to the stack. The end of the inner while loop can also be found if it

completes one full cycle of the used connections array without finding any node to remove.

Once the end of the path has been found, the algorithm will check the size of the stack, if there is nothing inside of the stack, the algorithm will stop. If there is something inside of the stack, the algorithm will use the top value to start its search for nodes to remove, and then remove it from the stack.

To find the sink nodes the algorithm uses the same method, but instead of using the first node in the connections, it will use the last node of the connections, and whenever the last node was used in this previous part, it will used the first node.

The next step of the algorithm is generating the height map for the graph. This is done similarly to the previous method, but with a few notable differences. These differences are in regards to output. In this version of the code, the algorithm needs to output where the rivers and rods are located on the terrain. These are found by first checking if the first node of the connection is a selected node, if it is the line drawing will also record the place that the rod should go. At the first iteration of the line draw algorithm, it records the x and y position of the river for the first side of the river. The x and y coordinates depend on how steep the line is. If the line is 45 degrees or more then the x coordinate is the line's x coordinate plus the river width for one side and minus the river width for the other side. The y coordinate remains the same. If the line is less than 45 degree, then it is the y coordinate which changes with the river width. The same is done for the last iteration of the line draw algorithm. The name of the river is also stored in an array, the name is simply the two nodes that it is connected too. If the line came from a selected node, then a rod needs to be placed. The location of the rod is the same as the line on a set iteration.

The rivers are then widened in the same way as before.
The program then needs to recalculate the shortest paths between the nodes, as it needs to output connections between the "new" nodes. This is done using a greedy algorithm, this algorithm will be given a starting river, then in a while loop it will first check if the end node of the river is one of the other "new" nodes, or if the end node is the bottom-right node. If either of these are true, then it will break the while loop. If they are both false, then the greedy algorithm will find the first river that connects to the end node, set the new river as the current river and then the while loop will start again, and it will carry on until it reaches an end node. The algorithm will then return the path it found to the next node.
Once all the new shortest paths have been found, these will be output to a file that will be used by the game, so that the game knows the route that the rivers take.
The next step of the program is determining which directions the rivers are flowing into a node, and flowing out of a node. This directions must be found as the game will use this information in order to know where not to place flowers on the terrain. This is done by looping through the new paths and extracting the start and end node from th paths. The algorithm will then search through the array and see if either the start node or the end node is one of the selected nodes. If the node is one of the new selected nodes, then it will check which direction the river came from by comparing the start and end node of the river to eachother. As each direction has a distinct amount of difference between them, the direction it is going can be determined. These directions are then output to a file for the game to read in.
At the end of this program the output will look like 6.9

Figure 6.9: Image showing the output without any randomly generated terrrain.

6.2.3 Terrain Generation

To randomly generate the terrain, the Diamond-square algorithm was used. This algorithm takes four values, to be used as the corners of the terrain, and then generates the rest of the terrain. it does this by alternating between the diamond step and the square step. The algorithm is a recursive algorithm, with each iteration running on a smaller grid.

The Diamond-square algorithm first takes in an input, which is the size of the step taken in each iteration, to begin with this value is the size of the terrain. This value is halved at each iteration.
The diamond steps in the algorithm are performed by iterating over all the possible y values in the grid, and at each iteration of that a for loop that iterates over all the possible x points is called. These for loops calculate every possible coordinate to perform the diamond step. The diamond step involves taking four values from the points that are at a distance of the size of the step taken away, and then averaging them and adding a random value. This step is shown in the second step and fourth step of 6.10.
The square step is performed by first calculating the possible values of the middle point of the squares. this is done in a for loop as the diamond step was. Then using the calculated coordinates, it will perform the square step on each coordinate. The square step involves taking the points at a distance of the size of the step taken in each of the cardinal directions. these points are then averaged and a random value is added to the value. This value is then placed in the height map at the coordinates that were found before. The square step is shown in the third and fifth step of 6.10.

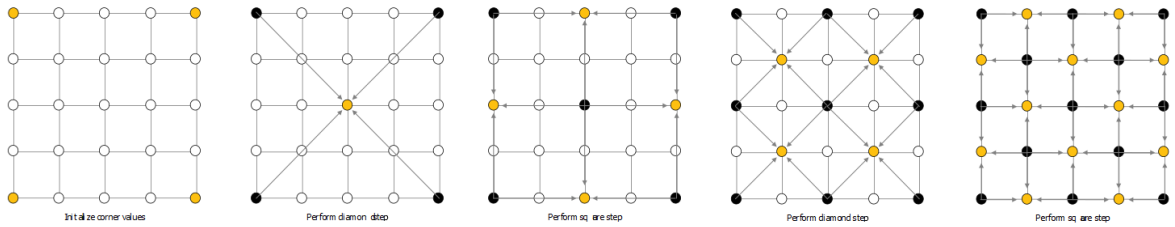

Figure 6.10: Image showing the steps of the Diamond-square algorithm. Image created by Christopher Ewin.

After the Diamond-square algorithm has been perform and has outputted the height map, the program will then use the heightmap output from the graph generation and the height map from diamond square and combine them into one height map. The equation for doing this is:

$$combinedHeightMap(x, y) = diamondSquare(x, y) - (0.05 * (1 - graphHeightMap(x, y)))$$

This formula was determined as the graph height map needs to be subtracted from the diamond square height map in order for the rivers to show up in the terrain. The graph height map is multiplied by 0.05 so that the rivers are not too deep on the terrain. The graph height map has to be subtracted from one as the rivers in the graph height map have the height 0, and the top of the terrain has the value of 1. This would mean if it is not subtracted from zero, the rivers would show up as mountains on the terrain.

6.2.4 Final Terrain

After these steps have all been performed the program will output the terrain as a model file, as well as many text files including data about where the rivers are, where the rods should be and the route the rivers take. When the model file is imported into the game it looks like 6.11.

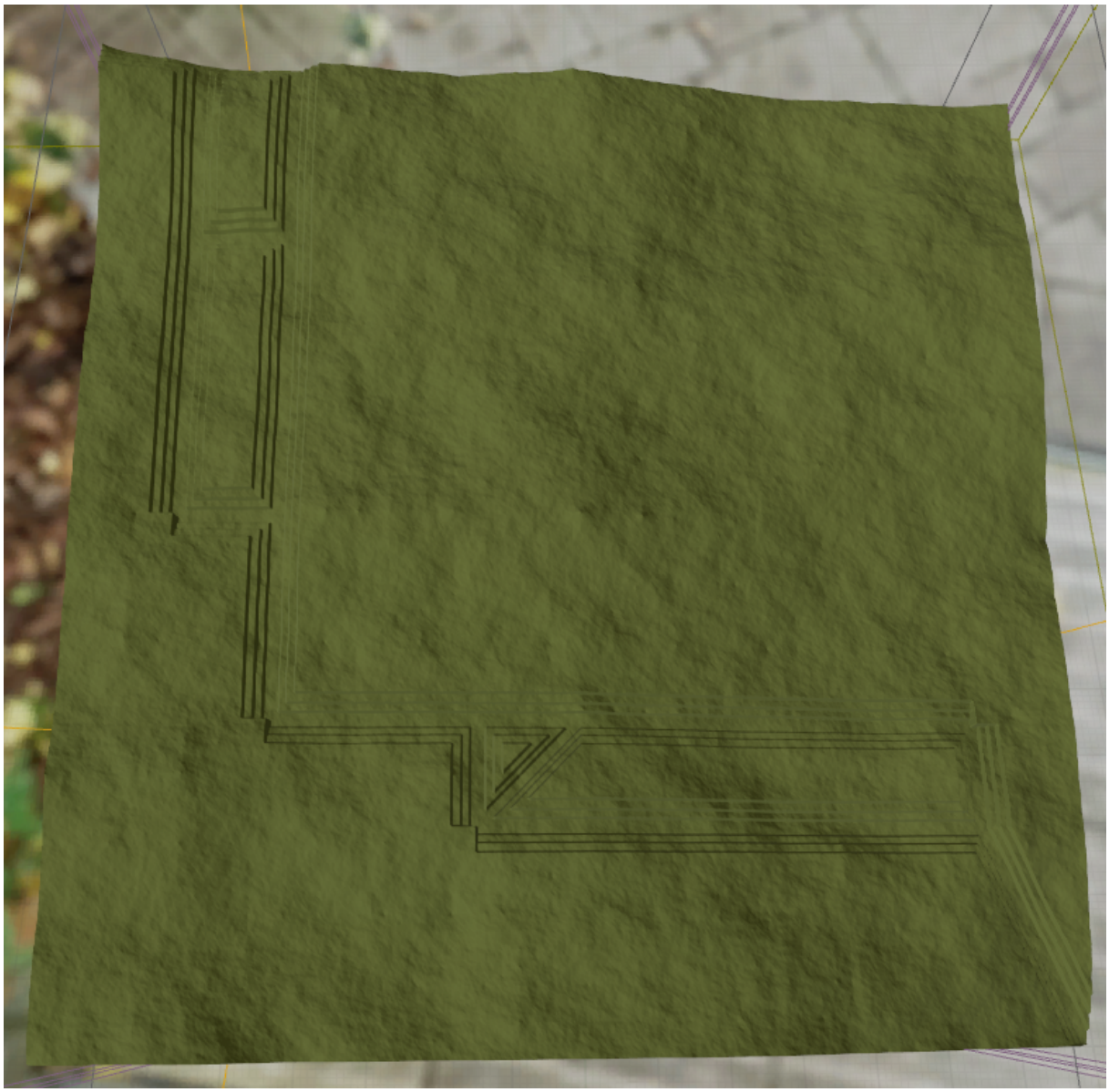

Figure 6.11: Image showing the final output of the terrain and river generation.

6.3 User Interactive Reverse Towers Of Hanoi

The logic for the reverse Towers of Hanoi was implemented by using collision boxes on the rod actor to act as trigger when Towers of Hanoi discs are placed and removed. This method and actor was used as the basis and control for the Towers of Hanoi logic. To differentiate between the discs, three different actors were created for each size of the Towers of Hanoi discs: small, medium and large.

A collision box covers the whole rod with a size that is bigger thane the size of the hole in each disc is set to BlockAll so a disc cannot be placed through the rod. When an actor comes in contact with the collision box, this triggers the rod actor's OnHit collision event. Within this event, it identifies the size of the actor through the name of the actor's class and does a check with the array of discs in which that specific rod currently contains. This simple check will only allow discs which are larger to be placed on top of smaller discs and also makes sure that the array doesn't add disc actors that are already in the array. When the check is passed, the collision box's collision profile name is set to OverlapAll to allow the disc that triggered the event to be placed in that rod otherwise it sets it back to BlockAll.

Furthermore, it adds an upward force to the disc to show that the action the player is trying to complete is invalid. This force is only added when the player is not holding the disc to avoid issues when the player is holding the disc and are moving/looking around and accidentally touching a rod with the disc.

A second collision box that is slightly larger than the first was used to handle the adding and removing of discs to the array which contains all the disc actors which the rod currently holds. When a disc overlaps this collision box it adds it to the array through the OnOverlapBegin event trigger then removes it from the array with the OnOverlapEnd event trigger. When a disc is added to the array, it sets all the discs below it to not allow the player to grab it. This is done to keep with the Towers of Hanoi logic of only the disc at the top of stack being movable.

A bonus feature to help with placing the discs was implemented with the 'snap to rod' feature. When a disc is allowed to be placed on the rod so the disc is larger than the disc below it, then as soon as the OnOverlapBegin collision event is triggered, the disc is teleported to the top of the rod with its rotation reset so it can slide down the rod. This is useful as it can sometimes be bothersome to the player to place the disc exactly so the rod can fit exactly through the hole of the disc. This feature is only in effect when the disc is not currently being held by the player to avoid the disc teleporting whilst the player is still holding the disc which causes the disc to still be attached to the player's hand even though the disc is not in range of the hand's grab sphere.

6.4 River Graph Flow

Graph flow logic was implemented by using information generated with the terrain such as the nodes, IDs of rivers and knowing which rivers are connected to which. An actor for each river is created with a mesh using the UProceduralMeshComponent and the vertices supplied in the rivers.txt file. Each river is given an ID with the first part of the 2 numbers of the ID as the node the river is connected to and the last two numbers as the other node the other end of the river is connected to. This means that a series of river connections have the last two numbers of a river ID identical to the first two numbers of the next river in the connection.

The river actors are placed in the game world by transforming the coordinates of the vertices for the river mesh generation. Since the coordinate system used for the terrain generation is different to the one used by the Unreal Engine, coordinates produced needs to be transformed so they are correctly placed in the game world as seen in Figure 6.12. The X axis is shown in red, Y in green and Z in blue. The top shows the difference in coordinate systems with the coordinate system used to generate the terrain and its output on the left and the Unreal coordinate system on the right. The bottom shows a top view of the terrain ignoring the Z axis since it is the same for both coordinate systems. (0,0) for the terrain file starts at the top left corner where as in Unreal the terrain mesh is placed at the middle of the world at (0, 0, 0). It shows an example of a terrain size used of (1024, 1024) and since the terrain mesh is placed at (0, 0) in Unreal, the centre of terrain becomes (0, 0).

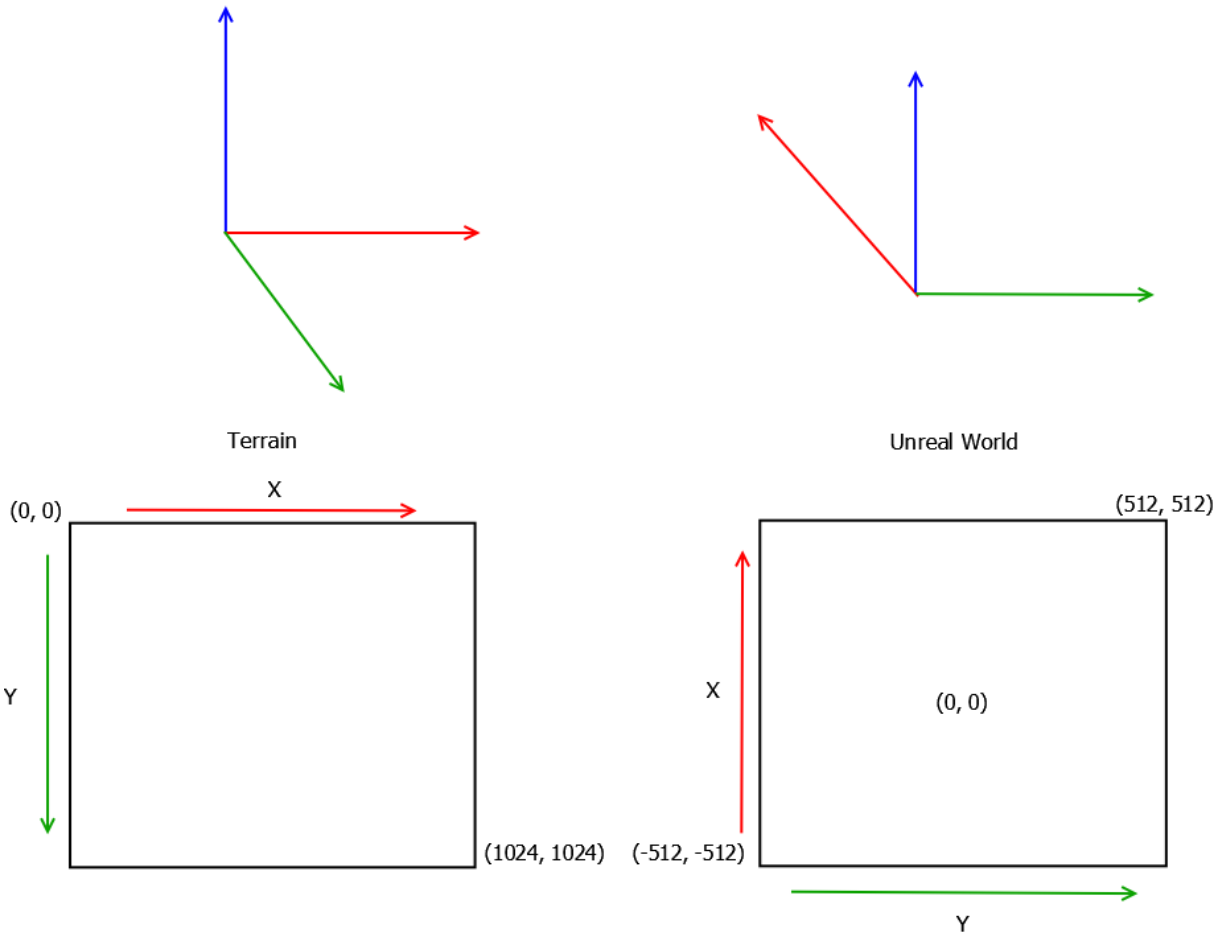

Figure 6.12: Image showing how terrain coordinates are different to Unreal Engine coordinate system

To calculate the transformation, a bounding box surrounding the terrain mesh is created as seen in Figure 6.13. The vertices of this box is used to compute the scale.

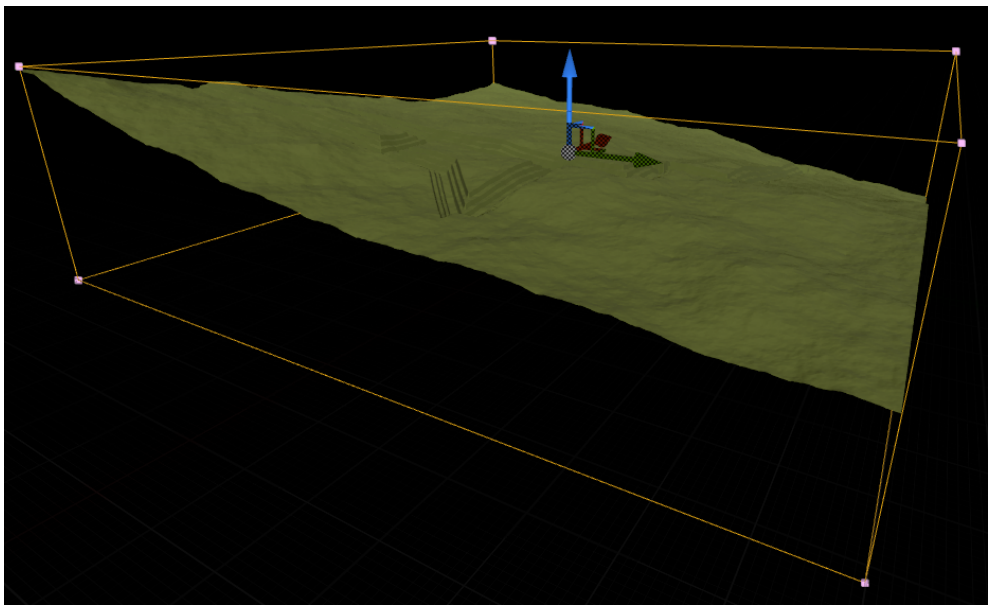

Figure 6.13: Image showing bounding box surrounding terrain mesh

The following formula is then used for the X and Y coordinates since they are of the same length due to the map being square:

$$transformedCoord = (coord - \frac{1}{2}terrainMax) * \frac{worldCoordMax}{terrainMax}$$

The value of *coord* is the coordinate given in the output file from the terrain generation using the origin coordinate system. $terrainMax$ is the max size of the terrain given by the terrain output so with the example in Figure 6.12, this would be 1024. Half of this is subtracted as the offset to make the coordinate in the same position but instead in the bottom left corner instead of top left to fit with the coordinate system. Now this is scaled by computing the scale factor using the bounding box coordinate of a vertex but doubled as $worldCoordMax$ to get the length of X (Y is the same) divided by $terrainMax$.

The Z transformation is similar but the only transformation needed is a scale which is just the height of the bounding box. This is computed by subtracting the Z coordinate of an upper corner of a bounding box with the Z coordinate of a lower corner. The $terrainMax$ is not used because it would always be 1 since the Z coordinates of the terrain are between 1 and 0 with 1 as the highest point and 0 as the lowest.

This method is then used to place the rods in their right places in the world so where there is a split in a river, a rod is placed on each of the rivers that branch out. The position of the rods are read in through the rods.txt file output from the terrain generation process. Each rod is given information regarding its own node ID and the river actor which it is connected to.

The source of the river in the graph has a flow value hard coded and then iterates through its river connection changing the flow value of each river then when it reaches the final river it means that the river has reached a node where it splits in to two. This split has a rod for each river in order for the player to change the flow of each river. With rods with no discs, the flow value is just split simply in half between the two rivers which then follows the same procedure as before and changes the flow value of the rivers in its river connections.

With the case of rods with discs in them, the flow is then affected depending on which discs are used. A small disc will reduce the flow value of the river by a $\frac{1}{6}$, medium disc by $\frac{1}{3}$ and a large disc by $\frac{1}{2}$. This means using all three discs would reduce the flow to 0 so completely blocking it off. Adding or removing a disc from a rod would change the flow value for the river the rod is connected to which then changes the flow value of all the rivers that this river affects.

In order for the player to easily distinguish a difference in the flow of a river when changing them with the discs, the opacity of the material used on the river will change depending on the flow value as a percentage of the original flow value at the source.

6.5 Flow Dependant Flora

Flower models are spawned around each of the nodes with rods. The first node with rods do not have flowers as the flows of the river that will act as input to this node are the same value as the flow at the source due to the fact that it has yet to be split. These flowers will act as indicators to the player whether or not they are getting closer to the goal.

The position of the flower models are computed by using the coordinates of the nodes found in the nodes.txt file output through the terrain generation procedure. The coordinates are then transformed to be placed correctly in the world with the same process as before. Another file is used called nodeConnections.txt which shows information on directions of the rivers that node is connected to. This is used to place the flowers in places on the terrain where there is no river. Due to method used to generate the terrain a river can go in one direction out of a possible 8. These possible directions can be described using the points in a compass e.g. North, North-East, East and so forth. Using random number generation, a flower has a one in three chance of placing a flower on a free space but makes sure at least one flower is placed for each node.

When the player places a disc on a rod and changes the flow of the corresponding river and all the rivers affected by the change, all the river flows that serve as input in a node are accumulated and checks against the required flow to comple the goal for the node. If the flow value is over the required flow, the material for the flower petals are changed to a blue colour. If the flow value is lower than the required flow, the petals then change to a white colour. When the required flow is met, the colour is changed to an orange colour. This visual indicator helps the player see their current progress and helps to solve the puzzle.

6.6 Game Goal Computation

In order for the game to provide a challenge to players such as University of Leeds applicants during applicant days, there needs to be a goal or a puzzle to be solved. The goal is implemented using random generation every time the terrain class is added in to the Unreal Editor scene. This randomness is controlled by limiting it to only possible solutions and this is done through simulation of the game before it is created.

For each rod actor in the map, one of the many possible disc combinations is chosen out of random which includes no disc on that rod. Then the game is simulated with the river flow values which are then affected by the discs on each rod. The flow value for all the rivers that act as input to the ndoes are then accumulated and is saved as the required flow for that node. The simulation is ended and resets the game with no discs on every rod then resets all the flow values to be computed again. The player will then need to try figure out the correct disc combinations for each rod through trial and error to meet all the required flows for each node to make the flower petals turn to orange which means the goal has been met. When all flowers turn to orange, the level puzzle is completed so the game transports the player to the next level.

Chapter 7

Testing and Evaluation

7.1 Testing Against Requirements

The original requirements that the client had specified were:

- Create a technology demonstration for the HTC Vive
- Make the technology demonstration appealing to students applying to the University of Leeds.
- The technology demonstration has to be appealing to members of graphics industry.

At least two out of three of these requirements have been met. A technology demonstration has been created for the HTC Vive, this has been shown and proven to the client. The technology demonstration has also been shown to be appealing to a member of the graphics industry, this was tested by showing the technology demonstration to a member from the graphics industry. The only requirement that has not been tested is if the demonstration is appealing to applying students, this has not been tested as there has been no opportunity to show the demonstration to an applying student.

7.2 Client Evaluation

The client has been shown the technology demonstration and is pleased with how the demonstration has progressed. The client was shown a final version of the product and had no major criticism about the demonstration.

7.3 Project Evaluation

7.3.1 Schedule

For this project the schedule was fairly accurate in terms of how long everything would take to do. There were three separate occasions when the schedule was not held to, but the project was back on track the week after. These three occasions were:

- The Virtual Reality features were not done by the week that they were supposed to be done. This was due to the fact that there was no room for the Vive to be set up in, and therefore the features could not be checked as the Vive could not be used.
- The Tower of Hanoi logic was also not finished in the time designated by the schedule. This was due to a bug being in the code that was hard to debug.
- Adding water to the graph had to be delayed compared to the schedule due to the fact that the original idea we had for the water flow (placing a flat sheet of water under the graph) would not work, and a more complex solution needed to be made. The original solution did not work as each river needs to have separate flow running through it and this is not possible with all the rivers being one sheet of water. This task was pushed back further in the schedule.

Most of the milestones for the project were met, only the first milestone was not met. This was due to the fact that a room for the Vive was not available until the week before the milestone, so the Virtual Reality features were not implemented properly.

7.3.2 Additional Features

No additional features were implemented in the project, this was due to the fact that the baseline features were sufficiently challenging to implement, and therefore no time was available to add the additional features.

7.3.3 Difficulties with the Project

One of the biggest difficulties with this project was the fact that only one Vive was available, and there was only one computer to develop on the Vive. This was solved by having each collaborator work on a separate part of the project, usually one would be working on the Vive using Unreal Engine, and the other would be work on the Out-of-Engine features, i.e. Randomly Generating Graphs. If the only feature that needed to be developed was in Unreal, then a pair programming approach would be used.

7.3.4 Meetings

Throughout the project we had weekly meetings with the client to discuss the progress of the project, and to show the current state of the project. These were immensely helpful to help keep the project in time with schedule, and they also helped with making sure each feature was up to the standard of the client. During the project the collaborators also had regular meetings in order to test the features if the demonstration, to make sure that everything worked as it should. During these meetings the collaborators would also discuss the plans for the week, and the tasks would be distributed out, and a short discussion would be had about the best way to tackle the tasks for that week.

Chapter 8

Conclusion

8.1 Conclusion

For this project, the problem that was undertaken was that the School of Computing at the University of Leeds have been using off the shelf demos for the HTC Vive virtual reality hardware. These demos are shown mainly during applicant days for potential University of Leeds students to spark interest in computing and technology. The problem was that there was no demo developed by University of Leeds students. Producing a solution for this would provide the School of Computing with software that can demonstrate what taking a computing course at University of Leeds can teach you and what the skills learnt can help you produce.

In order to provide a solution, background research as well as a demo requirements investigation with the client was conducted so the problem can be specified and narrowed down to a size where the project can be undertaken within the time constraints. This consisted of brainstorming possible ideas to what the technical demo could include whilst taking in to account the limitations and requirements such as short gameplay and related to the University of Leeds. A feasibility research was also needed to make sure that all hardware and other requirements can be met for development to happen.

A game was designed which included computer science aspects Towers of Hanoi and graph logic. The development of the game included random generation of terrain with randomly generated graphs as rivers using multiple algorithms such as the Diamond-square algorithm for the terrain. These are ouputted as a model file and txt files which have information where the rivers are, the nodes of the river graphs and the locations of the rods for the Towers of Hanoi logic. New terrain and river graphs can be generated easily by executing the terrain generation program again.

The game portion included developing reverse Towers of Hanoi logic which means discs that larger than the discs already on the rod can be placed on top. Players have the ability to navigate around the terrain using the motion controllers to teleport and also grab the discs and drop them on the rods. River flow is affected depending on the size of the discs on the rod and how many of them there are as they are blocking the flow of the river. Flowers are placed at each node in the river with required flow values and act as visual indicators for the game progress. The player must match all the required flows for each node to make the flowers' petals orange to complete the level and advance to the next one.

A skybox was created which used the Google Photo Sphere camera technology that takes multiple pictures and stitches them up to create a 360 degree picture. A cubemap was created with this to make the skybox that surrounds the player in the world. Skyboxes made with pictures taken around the University of Leeds campus was used for each of the different levels. This makes the demo meet the requirement of having the University of Leeds be part of it making it suited for demos during applicant days.

With the solution produced, it was concluded that the requirements set out to meet was achieved and the problem was solved. A technical demo was designed and developed that provided a puzzle game that uses computer science concepts and included the University of Leeds. The project was a success

and deadlines set in the plan were met consistently with very few minor delays.

8.2 Future Work

With the time constraint imposed, it was clear that further features could have been implemented to improve the gaming experience if more time was available and a few of these are outlined in this section.

One of the improvements that could be made is to develop better loading screens to transition between each level. With the current solution, the transition between levels is made with a simple delayed fade out to black then fade in to the next level. A loading screen which congratulates the player on completing the level would help the player know that they have completed the puzzle for that level. A title/loading screen to appear at the start of the game which provides information on the game such as instructions is another improvement that could be developed. Instructions can include how the puzzle game works and what the goal is as well as information on the game controls for teleporting and grabbing.

Another possible improvement is to improve the aesthetics of the game. This includes using or creating a proper water material for the rivers that is animated with river flow. The flow of the river will be more visible and a change of the flow with a disc change will be more evident to the player instead of just changing the opacity of the river material depending on the flow. Background scenery could also be improved which could be made with adding trees, plants and ambient creatures. The trees and plants can be generated using procedural generation with Lindemayer systems.

Optimisations can be made to improve the performance of the game. If high resolution map sizes were generated, it has a negative affect with the game performance due to the higher number of vertices that has to be read in to Unreal and rendered. This means the map resolution is limited, so optimisng this can lead to higher resolution maps. Another optimisation that could be implemented is with the use of culling. The terrain could be split in to smaller chunks and only render the terrain chunk and actors on that chunk that the player is currently on. This could mean scaling the map to a bigger size for more complex river graphs and a larger world.

A gameplay improvement that can be made is with the Towers of Hanoi discs. Currently, a set number of discs are placed around the world and the player must teleport and find these discs. This improvement is inspired by SteamVR's Vive tutorial which uses the different sections of the touchpad to spawn a different coloured balloon out of the motion controller. The game could take this idea and have the ability to spawn different sized discs using the motion controller which would make the game better for the player.

References

[1] Epic Games. About unreal engine 4.

[2] Epic Games. Engine features.

[3] Frontier Developments. Elite dangerous.

[4] Google. Google vr sdk for unity.

[5] C. Hoffman. The htc vive and valve's steamvr don't yet support linux and steamos.

[6] I-Illusions. Space pirate trainer.

[7] D. Parikh, N. Ahmed, and S. Stearns. An adaptive lattice algorithm for recursive filters. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 28(1):110–111, 1980.

[8] Valve. The lab.

[9] Valve. Steamvr.

[10] Valve Corp. Htc vive.

[11] Valve Corp. Steamvr tracking.

[12] C. Wheatstone. Contributions to the physiology of vision. part the frist. on some remarkable, and hitherto unobserved, phenonmena of binocolur vision. *Philosophical Transactions*, 128:371–394, 1838.

Appendices

Appendix A

External Material

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Appendix B

Ethical Issues Addressed