3D OpenGL Graphic Text Adventure

James Mitchell Flaherty
AC40001 Honours Project
BSc (Hons) Applied Computing
University of Dundee, 2018
Supervisor: Dr I. Martin

Abstract - *Interactive fiction is an old genre of video game that is no longer played outside a select community of dedicated individuals keeping the genre alive.*

The overall goal of this project is to create a type of interactive fiction game using OpenGL to create modern 3D graphics to help visualise the story that is told. The game developed allows the player to play through a story told through 3D environments and a text adventure inspired graphical user interface.

This document aims to describe the process of developing the game and how the game was finally implemented. It outlines all of the aspects of the development cycle starting with research; followed by the specification and requirements; design and prototyping; implementation and testing and finally the evaluation.

1 Introduction

The original aim of this project was to create an arcade 3D android game using OpenGL ES(Open Graphics Library for Embedded systems).

The initial idea for the type of gameplay that would be implemented included single-player narrative gameplay derived from the classic Interactive fiction genre. In short, the player will be given an initial cue for the scene and options on what to do. They will then be allowed to select an option of their choice and the story will advance and the scene will change in relation to the option that the player chooses.

The player will also be given a 3D scene with 3D objects and animations. The scene will change, and the players' position and viewpoint will change dependent on their decisions and their current position in the story.

During the background research and into the design process it became apparent that the type of game that the author was aiming to make was more suitable for a more advanced and powerful system like a console or PC. The atmospheric gameplay and general scope of the game would not work well on an android system.

After a few meetings the author and the supervisor of this paper decided to change the scope of the project. So, the new aim of this project was to create a 3D Adventure video game for the personal computer. This game should include features such as 3D objects, 3D Shaders, 3D lighting and some form of adventure gameplay mechanics.

For this, an environment that outputs 3D graphics is a vital component. Another component that is vital is the addition of a Graphical User Interface with user input via a keyboard.

For the software design process, the traditional Waterfall approach was adopted, to give a structured process to the project. This starts off with the research section where all of the necessary background reading is done and research into how to create graphical adventure games and text adventures. Prototyping is also then performed to learn more about 3D graphics programming and learn techniques needed to create the game.

Once all of the necessary research was completed a software specification was created as well as formal requirements. These requirements provided goals to meet for the end of the implementation stage.

Throughout the implementation stage the basic software structure was created. As well as all of the scenes and their resources were created alongside the story and the graphical user interface.

Once the implementation stage was complete the user tests were conducted that had users play through the current version of the game and evaluate it. This was helpful and informative and provided valuable feedback for any changes that were needed for the game.

The purpose of this document is to outline the process taken by the author when creating the game for this project.

2 Background

To create this game there were a few topics that were researched to help understand how to go about developing it. The covered topics consisted of components that with the research done into understanding them, would be implemented to help create the final product. This was a vital process that needed to be undertaken as it consisted of working with different 3D development environments such as OpenGL.

2.1 Understanding 3D Graphics

To begin with, research was undertaken into understanding 3D graphics. Three-dimensional computer graphics are images created using computers with the use of specialized graphical hardware and software. There are many different development environments that could be used to create applications with 3D graphics, a few of which were researched to find the correct one for this project.

2.1.1 OpenGL

OpenGL (Open Graphics Library) is the most widely adopted 2D and 3D graphics API (Application Programming Interface) in the industry. It is window-system and operating system independent, making it very versatile and enables developers in many different fields the ability to make detailed, streamlined and multifunctional graphics software applications. OpenGL is typically used with the GPU (Graphics processing Unit) and the latest graphics hardware to accelerate the rendering process.[1]

At the start of this project the author undertook the module "AC41001 & AC51008 - Graphics (18/19)" taught by the supervisor of this project. This module was taught using OpenGL and C++ and gave a thorough overview on how 3D Graphics work and how to use OpenGL to develop 3D graphics using objects and shaders. OpenGL was in the brief as a vital part of the project and recommended by the supervisor.

2.1.2 OpenTK (Open Tool Kit)

OpenTK (Open Toolkit) is a set of fast, low-level C# bindings for OpenGL, OpenGL ES and OpenAL. It translates OpenGL so that it can be coded in C# instead of the native C/C++. [2] It is as versatile as the core OpenGL as it mirrors the raw OpenGL API [3]. It provides several resources such as utility libraries such as a math package, its very own windowing system using Windows.Form, and an input handling system.

OpenTK was researched due to the author's familiarity with the C# programming language. After researching many positives would come with using OpenTK such as the ability to program in a more comfortable language and the inclusion of a native window and user input system would have been very useful.

Some negative aspects were also identified such as the time that would be needed to learn to use a completely new toolkit, there were less resources readily available to help in development, and both author and supervisor were unfamiliar with the toolkit.

2.1.3 Unity

Unity is a multi-platform, high performance, real time engine created by Unity Technologies. The unity editor boasts 2D and 3D development, an in-built physics engine, built in intuitive user interface system, and high-performance C# programming. [4]

The most important part of Unity in the context of this project is its real time graphics rendering engine. Its multiplatform fast native graphics API stays close to the low-level graphics API of the platform it is on and allows developers to take advantage of the latest GPU and graphics hardware improvements currently available. It also includes a post processing stack that allows developers to easily apply effects such as antialiasing, reflections, fog, and depth of field. There is also a Smart Camera System that allows for cinematic and intuitive camera controls that would be an asset to any project. [5]

Unity is another development environment that was researched due to the author's familiarity with C# as well as the editor itself. The author has previously developed a game using Unity and C# so has basic knowledge of its features and uses. Other notable positives Unity brings is it's in built UI and physics engine. Plus, it does not require a windowing system and games developed in Unity are highly portable as Unity allows deployment to multiple different systems. [6]

However, Unity shares some of the negatives that come with OpenTK as there is not as much resources readily available as there is for OpenGL and the project supervisor had limited knowledge on the environment and would not be able to as help as much if the project was developed using it.

2.1.4 Maya and Blender

Both Maya and Blender are professional 3D modelling software. They were used to create the 3D models and assets that are loaded into the game to create the scenes. These tools were used in place of constructing objects using OpenGL primitives via manual code. Doing the latter would be more computational heaving and increase load times in the game. This saved a lot of time in the development process as manually constructing objects using OpenGL primitives via code is more time consuming and difficult than creating objects using the aforementioned modelling tools.

2.2 Interactive Fiction

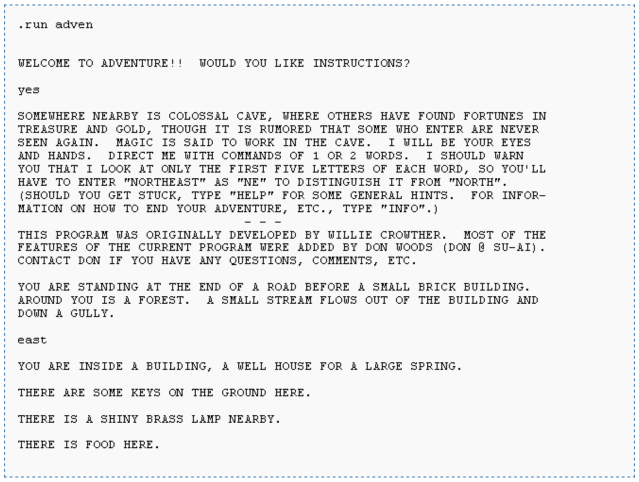
Interactive fiction is software and video games that allow the player to use text commands to manipulate characters and objects in a simulated environment that are conveyed through text. These games follow a non-linear narrative and allow the players choices to alter the plot through text input. Interactive Fiction games inform the player of the game state via a text output and the player uses text input to control the game.

The player is given a description of an environment and usually a cue to start off with and they use pre-defined commands that are usually read through a text parser and executed to interact with that environment. [7]

2.2.1 Examples of Interactive Fiction

Here are a few examples of classic interactive fiction games also referred to as Text Adventures:

2.2.1.1 Colossal Cave Adventure



```
.run adven


WELCOME TO ADVENTURE!!  WOULD YOU LIKE INSTRUCTIONS?

yes

SOMEWHERE NEARBY IS COLOSSAL CAVE, WHERE OTHERS HAVE FOUND FORTUNES IN
TREASURE AND GOLD, THOUGH IT IS RUMORED THAT SOME WHO ENTER ARE NEVER
SEEN AGAIN.  MAGIC IS SAID TO WORK IN THE CAVE.  I WILL BE YOUR EYES
AND HANDS.  DIRECT ME WITH COMMANDS OF 1 OR 2 WORDS.  I SHOULD WARN
YOU THAT I LOOK AT ONLY THE FIRST FIVE LETTERS OF EACH WORD, SO YOU'LL
HAVE TO ENTER "NORTHEAST" AS "NE" TO DISTINGUISH IT FROM "NORTH".
(SHOULD YOU GET STUCK, TYPE "HELP" FOR SOME GENERAL HINTS.  FOR INFOR-
MATION ON HOW TO END YOUR ADVENTURE, ETC., TYPE "INFO".)
                              - - -
THIS PROGRAM WAS ORIGINALLY DEVELOPED BY WILLIE CROWTHER.  MOST OF THE
FEATURES OF THE CURRENT PROGRAM WERE ADDED BY DON WOODS (DON @ SU-AI).
CONTACT DON IF YOU HAVE ANY QUESTIONS, COMMENTS, ETC.

YOU ARE STANDING AT THE END OF A ROAD BEFORE A SMALL BRICK BUILDING.
AROUND YOU IS A FOREST.  A SMALL STREAM FLOWS OUT OF THE BUILDING AND
DOWN A GULLY.

east

YOU ARE INSIDE A BUILDING, A WELL HOUSE FOR A LARGE SPRING.

THERE ARE SOME KEYS ON THE GROUND HERE.

THERE IS A SHINY BRASS LAMP NEARBY.

THERE IS FOOD HERE.
```

Figure 1 Screenshot of Colossal Cave Adventure

Colossal Cave Adventure was released in 1977 by Will Cowther and Don Woods and was the first known as the first ever text adventure video game, and also the first known work of interactive fiction.

The game had the player explore a cave via text commands to try and discover treasure and get out of the cave alive. The player would acquire points for completing these tasks and the games overall goal was to earn the maximum amount of points possible. *Figure 1* showcases the start of Colossal Cave Adventure, showing examples of the text commands used in the game.

This game had a simple premise, a well-structured adventure, and a well-designed map. It is considered to be the precursor to the adventure game genre as well as having great influence in the influx of interaction fiction games. A well-known game that was influenced by Colossal Cave Adventure is Zork.

2.2.1.2 Zork

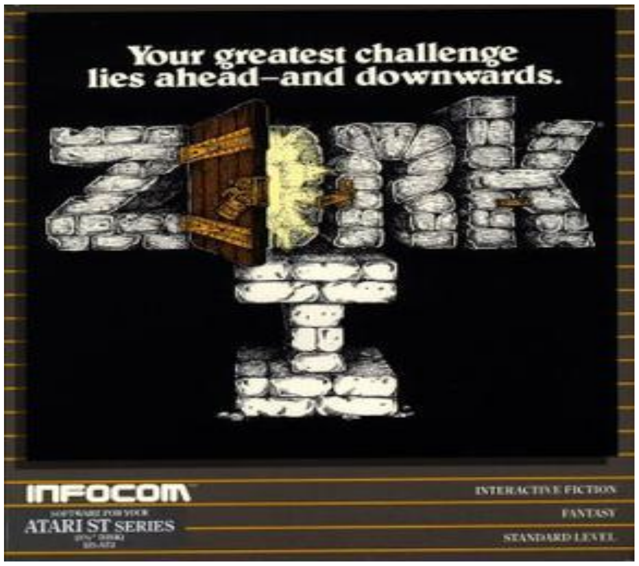

Your greatest challenge lies ahead–and downwards.
ZORK I
INFOCOM
SOFTWARE FOR YOUR ATARI ST SERIES
INTERACTIVE FICTION
FANTASY
STANDARD LEVEL

Figure 2 Zork 1 Box Art by Infocom

Zork was originally written between 1977 and 1979 by Tim Anderson, Marc Blank, Bruce Daniels, and Dave Lebling. Upon its public release, it was split up into three different games and released incrementally from 1980 to 1982. [9] The original boxart for ZORK can be seen illustrated in *Figure 2*.

Zork describes itself as a classic interactive fantasy set in a magical universe. The game places the player into a dangerous unknown land as a courageous treasure-hunter tasked with exploring this land in search for treasure and adventure. [10]

The objective of the game is to find and collect the treasures of the Great Underground Empire and put them in your trophy case. The player receives points - which lets them measure their progress - from solving puzzles, performing certain actions, visiting certain locations, and acquiring treasures. [10]

The game itself was developed around the same time as Colossal Cave Adventure but was definitely influenced by it. The developers spent a long time playing Colossal Cave Adventure even going as far as to hack the game to understand how it worked. [9] Zork made itself more memorable by being a much richer and fully realised game, with a large map, intricate puzzles and storyline. [11]

Throughout the years many different versions of Zork were developed and released. Some of these were compilations of the original games and others, sequels. Some of these were different and included images such as "Zork Quest:

Assault on Egreth Castle" which was an interactive computer comic book released in 1988. This games direction eventually led to the release of "Return to Zork" in 1993. "Return to Zork" was different from the rest of the Zork series as it was graphic adventure game which had video footage, detailed images and a point-and click interface.

2.3 Graphic Adventure Games

Graphic Adventure games are adventure games that have graphics in place of or in tandem with text. [12] Similar to Interactive Fiction, and traditional adventure games, the games in this genre typically have the player assume the role of a character in an interactive story which is driven by exploration, puzzle solving, and the player's decision. Using graphics to convey the environment to the player and a variety of input methods such as point and click, and the traditional text parser interface, graphic adventure games became quite popular and furthered the adventure game genre and produced many great and influential games.

2.3.1 Myst

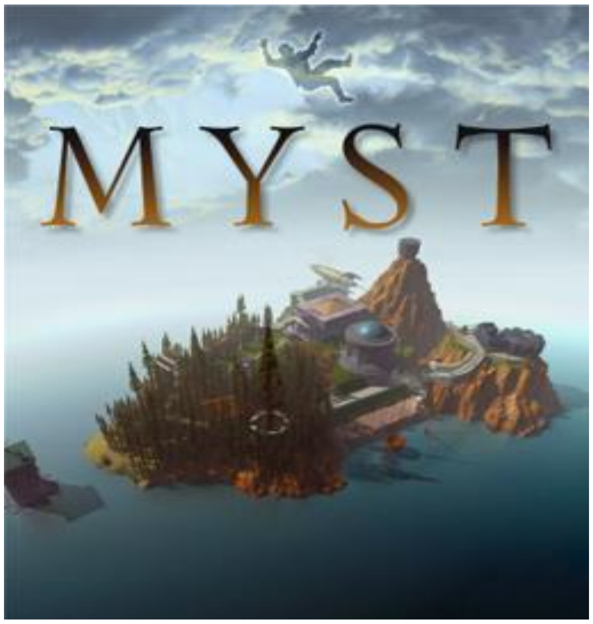


Figure 3: MYST cover art by Cyan Worlds and Brøderbund.

Myst was designed by Robyn Charles Miller and Rand Miller and originally released in 1993. It is classed as a graphic adventure puzzle video game and takes cues from the interactive fiction genre.

In Myst the player is immersed in a fictional world where they find themselves transported to the mysterious Myst Island where they must solve puzzles to learn more about the games characters. [13] The island that the player explorers in Myst can be seen in *Figure 3* in the illustrated cover art for the game.

The gameplay consisted of a graphic environment that was interactable via point and click functionality. The player could move around by clicking where they want to go and pick up and manipulate objects and puzzles by clicking on them.

Myst was highly successful at the time of its release. The graphics were mesmerizing for its time, the live action sequences were stunning, and the musical score was captivating. [14] Myst's 3D and interactive world was a big influence on this project as it was the gameplay and environment that was envisioned when the idea for the game was conceived.

2.4 Graphic Adventure Game Development

After researching graphic adventure games and interactive fiction games it became apparent that neither of these were entirely suitable for the original idea for the game. Though both of these sub-genres weren't completely suitable the Adventure Game genre that encompasses them was general enough to be suitable for this project. So work went into developing a type of hybrid of these game sub-genres under the general Adventure Game genre. First research must be done in the design and development of these games.

2.4.1 Design Patterns

Software Design patterns are very useful, general repeatable solutions to commonly prevailing problems in software design. These are not polished and directly translatable solutions but a set of instructions or a template for how to solve problems in many different situations.

Implementing design patterns in this project will speed up the development process as well as making the code of the project more structured and readable. There are a few different categories design patterns can be put into, for this project a behavioural, sequencing, and game design patterns were researched. [16]

2.4.1.1 State

The State pattern is behavioural design pattern that takes an object and assigns a state to it. When the state changes it allows the objects behaviour to change. Problems this pattern strives to solve problems such as changing an objects behaviour dependent on its internal state changes, and the addition of new states should not affect the behaviour of already created states. [17]

This pattern could be used in interactive fiction in an interesting way. Each room can be a state containing things such as objects and events, and the current state is the room the player is currently in. This is an interesting concept that would work well for this project.

2.4.1.2 Game Loop

The Game Loop pattern is a sequencing design pattern. The idea behind it is that a game loop is ran continuously during gameplay. The game loop consists of processing the user's inputs, updating the game state in conjunction with the user's inputs and any over variables that make a difference, and will also render the game itself. [18]

Due to using OpenGL the Game Loop design pattern will most likely be in use for this project. This is still good to note as the pattern works well for this game as the environment the player is put in is to be interactive and alive, so the game loop pattern is a perfect fit as it can be used to update the environments animations.

2.4.1.3 Eurekon

The Eurekon Design Pattern is a game design pattern found exclusively in expressive story games such as text adventures. The main aim of Eurekon is to have the player have some sort of realization when trying to solve a puzzle, which then in turn allows them to solve said puzzle in an adventure game.

There are a set of steps that are critical in this pattern, they include:

Step 1: The player is unsure that a random action will advance the story

Step 2: They then have a moment of realization, in which they realise that a previous random action might be the solution to their current problem.

Step 3: They then plan out what they will do and execute it feeling like they will be successful.

Step 4: The plan is successful; the player is then satisfied and relieved.
[19]

This pattern would be a good addition to this project as the game should be satisfying to the player and this pattern will help ensure that. To do this the narrative side of the game showed be well structured.

2.4.2 Interactive Fiction Development Technique

As a newcomer to creating a piece of Interactive Fiction there are a few rules that can be followed recommended by Dan Cox in his piece "Interactive Fiction: Text Adventures" [15].

These rules are focused primarily on traditional Text Adventures but for the purpose of this project something will be omitted or changed to fit the project better:

Step 1: To start off with, a story should be constructed that contains ideas for places, people and problems that occur with both of these. This will provide a framework for the game.

Step 2: Next pull out all of the characters in the story. These characters can then have all of their actions and information noted. Here characters represent everything from the player, objects and NPC's (Non-Player Characters).

Step 3: Using this list, a small sample list of commands can then be created and linked to the actions and information for each character. These include certain commands that directly refer to a specific point in the story such as "Open Chest".

Step 4: Now the next step is to take these lists of characters, commands, and the story to then plot the scenes. Plotting the scenes consists of the environment, what objects have to be there and what happens in them to progress to the next one. Here the start and end for the game is defined via the scenes.

Step 5: The last step is to start writing the game from a certain point and build it up sequentially. Place characters and objects in the right place and at the right time in the timeline of the story.

These altered steps were a good starting point for the project. They were adopted to make sure that the Interactive Fiction aspect of the game was still considered throughout development.

2.4.3 Target Audience

For the target audience, this project is an amalgamation of a few different genres and gameplay mechanics. So the target audience for this game is wide but also rather specific.

An audience that this project hopes to cater to, is the people - most likely older adults – who used to play the original classic text adventures.

Another audience that this project hopes to attract is atmospheric graphical adventure game fans, such as those who play Myst and other games like it.

Lastly a newer audience that this project relates to is the young adults that enjoy newer works of interactive fiction games such as episodic adventure games.

2.5 Compiling Research

To be able to complete this project a lot of different elements were required. The background research on 3D graphics programming helped when developing the games environment and functions. Additionally, looking into the adventure game genre gave a concise idea on how to structure the story successfully. Furthermore, researching design patterns both help create solid foundations for the story and the program structure.

3 Specification

Once all of the necessary background research on all of the vital components for the project was concluded, a project specification was developed. This specification was built from the foundations of the initial idea for the game that was introduced in the introduction and further expanded throughout the design and implementation process. It was also advised and refined up during meetings between the author and project supervisor throughout the entirety of the project.

For the specification, it was broken up into individual parts and a Gantt chart was created. This Gantt chart helpfully allows for time to be allocated to each part of the project and allowed to see if the project was behind in progress or not. Below in *Figure 4 Initial Gantt Chart* a small version of the Gantt chart can be found. A full-sized version can be found in the appendices.

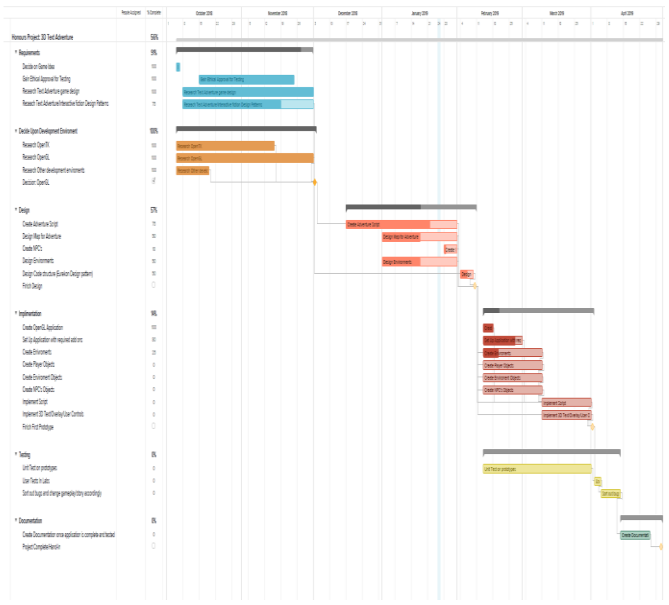


Figure 4 Initial Gantt Chart

3.1 Clear The Sky Specification

Figure 5 Clear The Sky Logo Mock-up

3.1.1 Gameplay

Overall the gameplay specification for Clear the Sky consist of aspects influenced from classic text adventure as well as games that preceded them graphical adventure games. A mock up of the game logo can be found in *Figure 5 Clear The Sky Logo Mock-up* above.

The player should start the game and be given an opening narrative to introduce them to the world in which they find themselves. This narrative will explain the player's predicament, explain their goal and provide a hint to get them started.

The player will then be given four options on what they can do at their current point in the narrative. These options will be linked to keyboard buttons which can be pressed.

Player's choices will either be the right one and advance the narrative or they may be wrong and most of the time will have no consequence except in some circumstance where their choice will result in the player having to restart the game from the beginning.

The aim for the player is to "clear the sky", hence the title. Basically, the player must make all of the right choices to advance the narrative and complete the story set before them.

3.1.2 Scenes

The game will consist of multiple 3D scenes in which the player will have to make all of the right choices to be able to move on to the next scene. This version of the project consists of 4 different scenes. These scenes will consist of different objects and terrains that change in conjunction to the narrative when the player makes certain choices.

3.1.3 Skybox

A must for these 3D scenes is a way to have a backdrop to make the game more immersive and atmospheric. To do this a set of skyboxes are essential. A skybox is a cube that surrounds the viewpoint of the player's camera and rendered behind everything else in the environment. This cube will have a detailed texture of a landscape or skyscape. The desired effect would be to give the players view a 3D panorama which is viewable anywhere the players view may point. There will be a few different skyboxes implemented as it is a vital mechanic for the game.

3.1.4 Object Loading

Another essential component needed for these scenes is the inclusion of objects. These objects will be placed throughout the environment and will be key for the narrative. To create these objects, both Maya and Blender will be used to create objects such as the pedestal and trees. Other objects are created using OpenGL primitives such as the chest as it consists of moving parts.

3.1.5 Sound Effects & Music

An essential part to the game is the sound effects and music. These are added into the game via the Audiere C++ library that is used in this project. Most of the music and sound effects are sourced online and credited in the game. The music and sound effects are used to add suspense and immerse the player in the atmospheric world of the game.

3.1.6 Graphical User Interface

For this game being a hybrid of a text adventure and a graphic adventure game, the players GUI is of utmost importance. The GUI must be simple but heavily text based. It must also blend in with the scene and not take away or block the 3D graphics.

The GUI is simple and takes up on a small percentage of the screen, is slightly opaque and has white bright writing to bring attention to it. The current story/narrative point is at the top and all the options available to the user clearly marked out. There is also a timer for certain events. There is also a death and loading screen that are an extension of the GUI but its only purpose is to take the player out of the game for a second, so they take up the whole screen.

3.2 Formal Requirements

Upon the completion of the specification, it can then be used to create a formal list of requirements. These requirements give a streamlined and concise list of functionalities for the game. This is then used to make sure all parts of the specification are incorporated into the final implementation. These formal requirements are as follows:

1. Game Development Environment
   - 1.1 The game must be developed using the C++ programming language.
   - 1.2 The game must be developed using OpenGL (Open Graphics Library).

2. Display, Control & Audio
   - 2.1 The Game must be controlled via a keyboard
   - 2.2 The Game must be played on a computer.
   - 2.3 The Game must have a visible Graphical User Interface.
     - 2.3.1 The GUI must have a description of the current state of the story.
     - 2.3.2 The GUI must have all of the player's current options.
     - 2.3.3 The GUI must have a Timer when it is required.
   - 2.4 The Game must include music and sound effects.

3. Scenes and Content
   - 3.1 The Game must include 4 scenes.
   - 3.2 Each Scene in the Game must include Objects.
   - 3.3 The Game must allow for the player to advance through all scenes sequentially.
   - 3.4

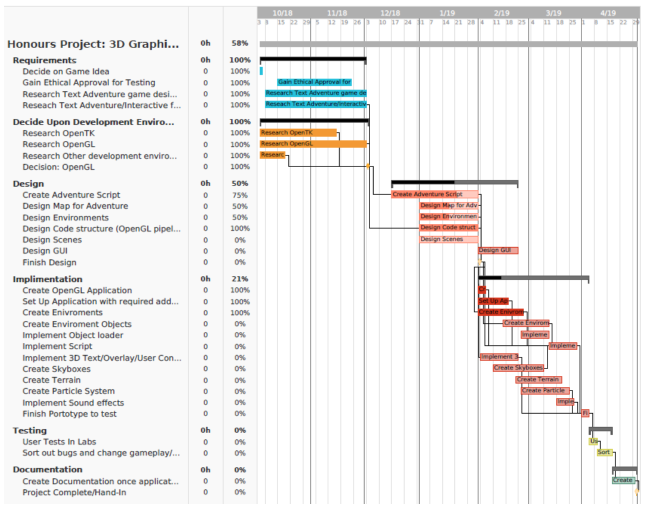
4. Story
   - 4.1 The Game must tell a story.
   - 4.2 The Story must have a start, and end point.

4 Design

4.1 Project Plan and Management

Before starting this project, a project plan is needed to make sure the project is completed on time and for the right deadlines. These deadlines included the Ethical Approval submission on the 9th of November 2018, The Mid-Project Progress Report due the 25th of January 2019, the information required for the degree show website/catalogue on the 12th of April, The final project portfolio submission on the 30th of April, and then finally the project demonstrations.

To plan and manage this project so that it was completed on time for the deadlines explained before, the Gantt chart that was shown before was revised to show the deadlines and updated accordingly throughout the project. The revised Gantt chart can be found in *Figure 6: Revised Gantt Chart* below and a full sized version in the appendices.



Honours Project: 3D Graphi...	**0h**	**58%**
Requirements	**0h**	**100%**
Decide on Game Idea	0	100%
Gain Ethical Approval for Testing	0	100%
Research Text Adventure game desi...	0	100%
Reseach Text Adventure/Interactive f...	0	100%
Decide Upon Development Enviro...	**0h**	**100%**
Research OpenTK	0	100%
Research OpenGL	0	100%
Research Other development enviro...	0	100%
Decision: OpenGL	0	100%
Design	**0h**	**50%**
Create Adventure Script	0	75%
Design Map for Adventure	0	50%
Design Environments	0	50%
Design Code structure (OpenGL pipel...	0	100%
Design Scenes	0	0%
Design GUI	0	0%
Finish Design	0	0%
Implimentation	**0h**	**21%**
Create OpenGL Application	0	100%
Set Up Application with required add...	0	100%
Create Enivroments	0	100%
Create Enviroment Objects	0	0%
Implement Object loader	0	0%
Implement Script	0	0%
Implement 3D Text/Overlay/User Con...	0	0%
Create Skyboxes	0	0%
Create Terrain	0	0%
Create Particle System	0	0%
Implement Sound effects	0	0%
Finish Portotype to test	0	0%
Testing	**0h**	**0%**
User Tests In Labs	0	0%
Sort out bugs and change gameplay/...	0	0%
Documentation	**0h**	**0%**
Create Documentation once applicat...	0	0%
Project Complete/Hand-In	0	0%

Figure 6: Revised Gantt Chart

4.2 Waterfall Approach

The traditional Waterfall software development approach was adopted for the development of this project. The project started with an in-depth research section that started at a low level and got more in depth and resulted in the creation of the software specifications and formal requirements. At this point prototypes were made to investigate which development environment would be chosen for the implementation stage.

Additionally, the design stage was undertaken where the software architecture was designed with the help of design patterns to fulfil the software specification and requirements. In this stage the story structure for the game and each of the scenes mentioned in the specification were designed as well.

After all of the required components were sufficiently designed the implementation stage begun. In the implementation stage the goal was to complete the project requirements. By the end of this stage a prototype was created that fulfilled the project specification. This prototype was then used in the next stage of development. The next stage was the testing stage. In this stage formal user tests were conducted on the prototype. From the user tests, feedback was given and was then compiled into [testing form]. Using this [testing form] changes were made to the prototype and helped with the final design of the game. This [testing form] allowed for a user centred design to be adopted and ensured all of the feedback from the tests were considered.

The Waterfall approach was embraced for this project as prior to undertaking the project, the author had little to no experience with core 3D programming as well as with creating narrative based adventure games. Thus, a lot of research and design was needed to get familiar with relevant technologies and techniques to effectively complete this project. Furthermore, this approach made the project more structured and helped provide solid documentation.

4.3 Prototyping

During the research stage of the project a prototyping phase took place to familiarise the author with 3D programming as well as to decide upon a definitive development environment for the project. During this phase time was taken to understand 3d graphics programming as well as to research and gather useful functions and techniques which would be used during the implementation stage.

4.3.1 Prototype 1: OpenTK Enviroment Prototype

For the first prototype the goal was to set up a environment in OpenTK that would allow for further development. OpenTK was the first prototype because it was found to be more suitable for the author as it was programmable using C# which the author was more confident and familiar with than with C/C++.

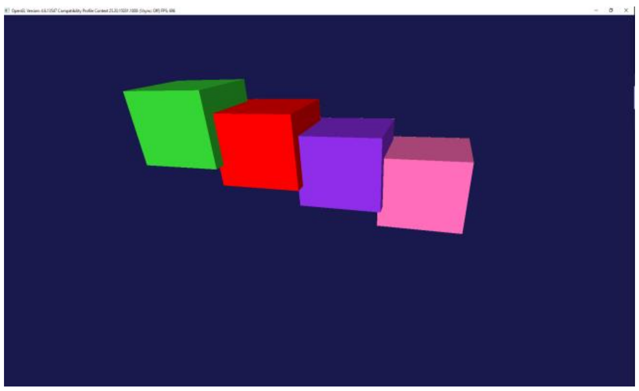


Figure 7: Screenshot of OpenTK prototype

In this prototype a window forms application was created in which an array of cubes was rendered using OpenTK which could be rotated and scaled using key commands on the keyboard. These cubes were created using OpenGL primitives that were then sent through to the correct buffers. Once the cubes were created the next step was attempting to add lighting to the scene. This proved difficult in this prototype as the way the cubes were rendered made it difficult to assign normals to each polygon in each cube accurate. This resulted in errors which couldn't be easily solved at the time as neither the author nor the supervisor was familiar enough with OpenTK to effectively debug the problem. An example of this prototype can be found in *Figure 7: Screenshot of OpenTK prototype* above.

4.3.2 Prototype 2: OpenGL Environment Prototype

For the next prototype, the goal was to set up an OpenGL application environment for testing and learning the API and 3D graphics programming. OpenGL was used for this prototype as it is one of the oldest libraries available – so it has a wide range of resources available to help with development.

In this prototype the GLFW windowing system was implemented to create an environment to render OpenGL primitives within. Once that was complete OpenGL primitives were rendered. First a simple 2D triangle was made. Once that was complete a 3D cube was then rendered made up of triangles, this task was undertaken to learn the fundamentals of creating objects in a 3D space and was completed successfully. Other objects were then implemented using the same method, including: a sphere, a rhombohedron, and two 5 faced cubes.

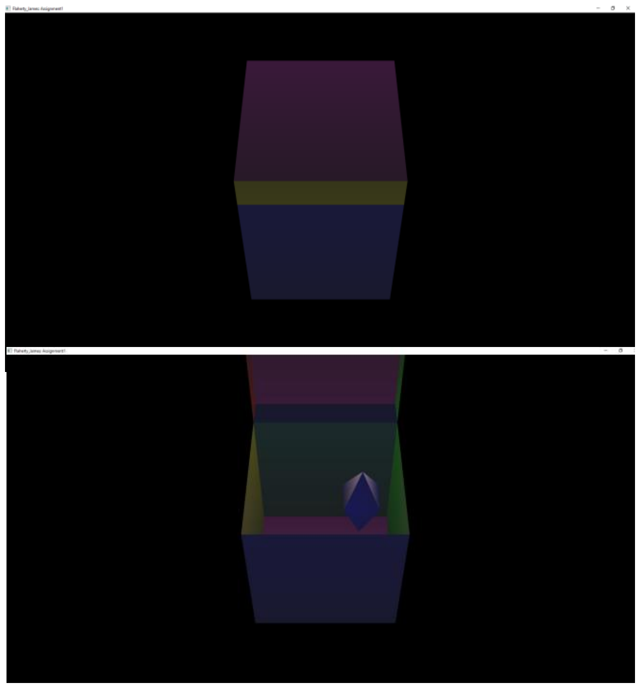


Figure 8: Prototype 1: Screenshot of Box open and closed with gem inside

Due to having more support when it came to OpenGL, normals were easily assigned allowing for lighting to then successfully be implemented via shaders. The last part added to this prototype was animation. In *Figure 8* above the before and after effects of the animation are visualised.

The finished prototype consisted of a chest made up of two 5 faced cubes, with one of the cubes moving via an animation which looked like the chest was opening from the top. A gem then floated out and rotated out of the box with a light flying around which was represented by a white sphere.

The results from this prototype were very strong and helped build solid foundations in the authors understanding of 3D graphics programming. It covered making an OpenGL context to draw on the screen, how matrices are used to create 3D images, and how shaders can be used to implement different types of lighting using different algorithms. All of these techniques helped when in the implementation stage. The finished prototype can be seen in *Figure 9* below.

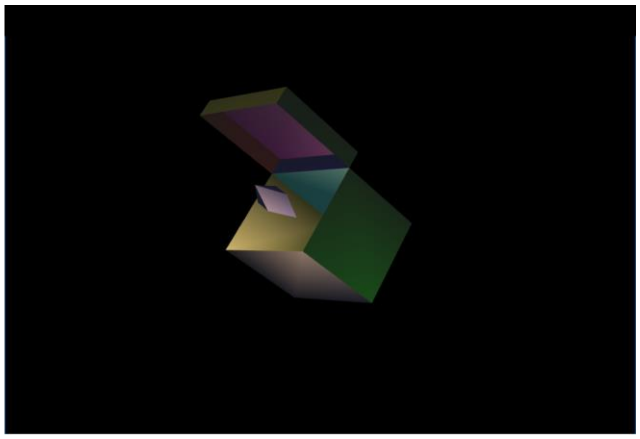


Figure 9: Prototype 1: Box rotated, open and gem with lighting

4.3.3 Prototype 3: Advanced OpenGL Prototype

This prototype is an extension of the second prototype. It built on the OpenGL environment and techniques that were already implemented and added more complex and advanced techniques.

The scene's focal point is the chest from the previous prototype, but instead of being in a black background, the chest is placed on top of a grassy terrain and surrounded by a gloomy backdrop of water and cloudy skies with some light breaking through. There are also dust particles that are meant to be seen when the light hits them. In *Figure 10* on the next page*,* the main focal point is shown as well as a couple other shots of other views of the prototyping showcasing the features mentioned above.

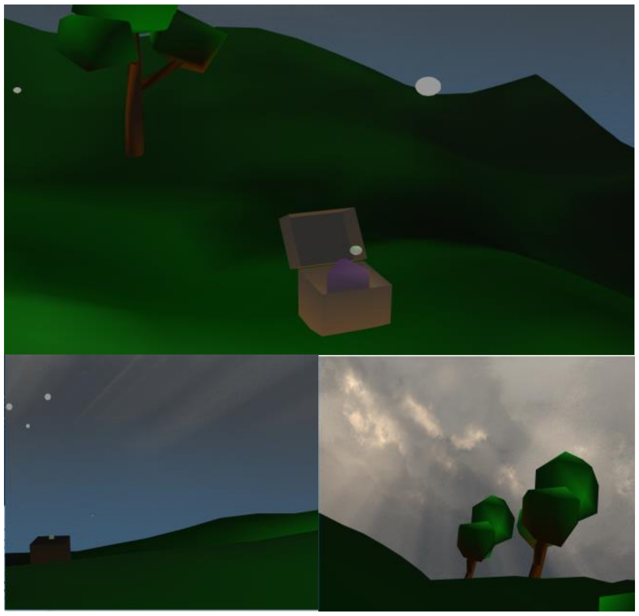


Figure 10: Prototype 3: Screenshot showcasing all advanced OpenGL techniques.

One of the first advanced techniques that was implemented was a realistic terrain function. Realistic terrain is created using Perlin noise which uses a series of random gradients that are predefined, that are then used to create a continuous function by interpolation that are all the different frequencies summed together. To do this the first step is to create a flat terrain using a grid of vertices and triangle strips. Next a two-dimensional array of noise values is made which is scaled to a pre-defined height range. This array is then used to set the height values for each vertex in the terrain. Then the normal and colours are calculated to be sent to the shader. This was an invaluable asset as it was an important component in the final component.

Other techniques that were implemented was a skybox which encompassed the terrain to try and immerse the user to make them feel like they are another place. There is also an object loader implemented to load in objects made in blender, this includes low poly trees and a gem. There was also fog added. Lastly a particle effect system was implemented to mimic dust particles, but problems were met with this.

Due to the set up of shaders in the code it was difficult to implement multiple lights in this prototype which made the desired particle effects hard to get correct. The particles were originally very small which made them very hard to see, especially from still images. So, for presentation purposes they have been enlarged to be perceivable in *Figure 11* below.

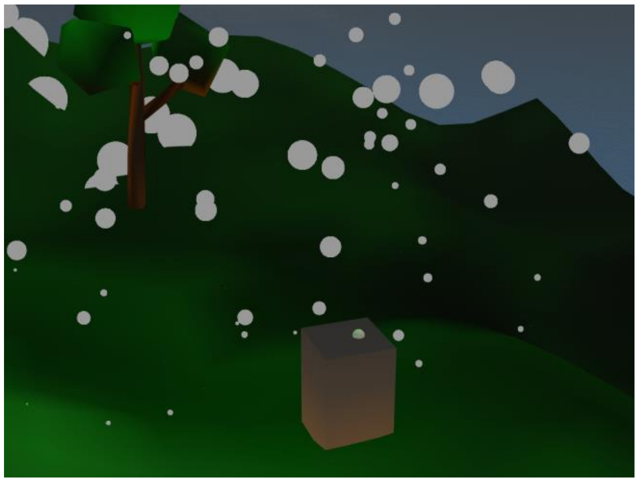


Figure 11: Prototype 3: Exaggerated Particle Effect

Another problem encountered during this prototype was problems loading in textures and placing them on objects. The concept was difficult for the author to grasp and help was required to be able to implement it.

Overall this prototype helped consolidate understanding of 3D graphics and the learning of a plethora of advanced OpenGL techniques. Some of these techniques and their implementation needed further research and practice to be able to implement them effectively.

4.4 Development Environment

Before the implementation stage could begin, the development environment has to be decided upon. Even though the brief stated the use of OpenGL ES originally, after consultation between the respected parties, the author was given choice of whatever development environment they wished. So, research was conducted on OpenTK, OpenGL and Unity to work out which one would be the most suitable one.

After the research had concluded, it was decided that the development environment that would be used was OpenGL. OpenGL will be used to create a desktop application using C and C++.

This conclusion was drawn due to Unity being too high level for this project. Unity's games engine was covered up by a lot of high-level functionality and this project required the author to work directly with the engine itself and build up.

OpenTK was heavily considered also due to the author's C# familiarity as well as it having compatibility with window forms applications. However, struggles with setting up an environment properly in the prototyping phase dissuaded the author. It was ultimately decided to abandon using OpenTK as development was taking too long due to unfamiliarity for both the author and supervisor and lack of sufficient resources readily available for it.

Thereafter, OpenGL was concluded to be the most suitable environment as the prototypes were much more successful in a fraction of the time. Other factors included it being platform and windowing system independent, and out with the project the author was also taking the module “AC41001 & AC51008 - Graphics (18/19)” which taught 3D graphics programming using OpenGL and C/C++. This contributed a significant factor to deciding which development environment as there were a lot of sources readily available to the author as well as the project supervisor being well versed on the topic.

4.5 Software Design

For the software design, a class diagram was created to be able visualize the intended structure for the game. *Figure 12* shows a cropped version of the structure and the original full-sized file can be found {Appendix}.

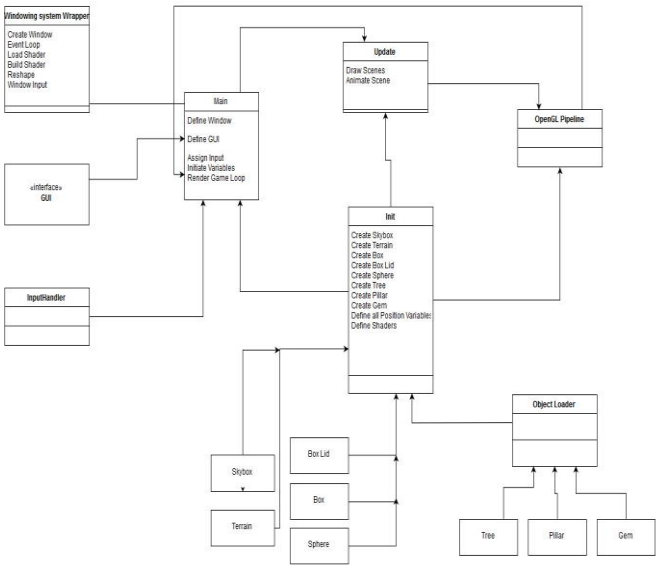


Figure 12: Cropped class diagram

In the class diagram, the Main class is the central class for this piece of software. The Main class calls all of the other classes and executes them in the order needed. It works together with the windowing system to create a window and link all other functionality to it such as the GUI, the Input System, and receives the 3D graphics output of the OpenGL pipeline and displays it in the view it creates.

The other important classes include the Init class which’s purpose is to set all of the variables needed to run the program as well as creating all the resources needed for run time such as the terrain and skybox. The object loader loads in object files created using Maya or blender and is called inside the Init to load in Tree’s, a pillar and a Gem. These to classes are the ones that are executed first by the Main function as they are critical for the other classes.

One of the most critical classes shown in the diagram above is the Update class. In this class the current scene and all of the related objects are drawn and updated every frame. It handles the movement and animations that happen every frame as well. Update is called in the Main class every frame and so is the Input handler and then the game is rendered through the OpenGL pipeline. These three main classes are designed using the Game Loop software design pattern which was covered in the research section of this project. In *Figure 13* below there is a diagram which illustrates the game loop pattern in the context of this software.

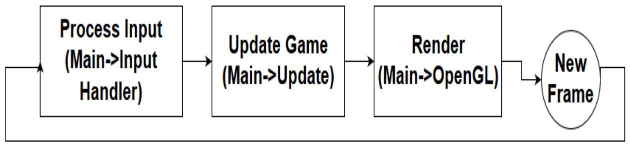

Process Input (Main->Input Handler)
Update Game (Main->Update)
Render (Main->OpenGL)
New Frame

Figure 13: Game Loop Design pattern visualisation

The use of these diagrams helped quite a lot when starting off writing the code for this project. The use of design patterns helped build solid foundations for the game.

4.6 Story and Scene Design

Once the games structure was designed the next step was to create an outline of the story. To do this, the Eurekon design pattern was used as a foundation to help build a simple story to get the game started.

Before the story could be finalised the scenes had to be designed. It was decided to create 4 different scenes that share the same objects and similar terrain. The story was then designed so that the player’s actions will trigger change in the terrain and objects effectively changing the scene. The player will be able to trigger these changes when they come to the conclusion on what the correct action is for them to take. These changes to the scene are also designed to award the player for choosing the right action and give them a sense of satisfaction.

In *Figure 14* below, is a diagram of the scene design and the connection between them. This was originally drawn on paper but digitised for the purpose of this paper.

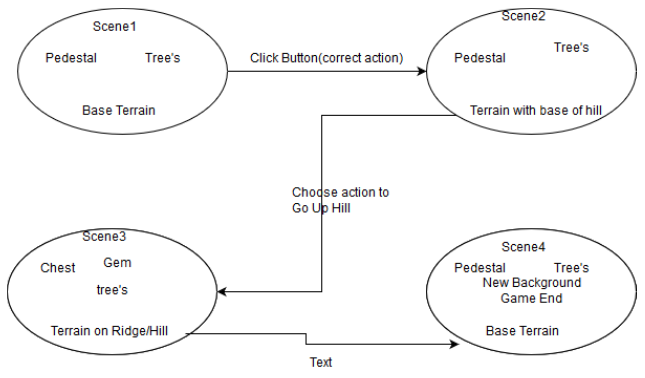


Figure 14: Scene Design

4.7 Graphical User Interface Design

One of the last parts of the game that needed to be designed was the GUI. To design this the Anttweakbar OpenGL library was used.

The GUI that was designed is using the Menu-Driven Actions method [20]. With this method the game has a GUI that displays options and allows the player to select their action from this menu of options. The aim of this method is to give the player a clear choice of their choices though it does take away from the immersion of the world they are in.

Even though this method of interface design negatively impacted the player's immersion it also effectively simulated a text adventure interface which was what was also what was desired. So, using this, and the specification for the GUI defined earlier the design for the GUI was finalised.

4.8 Testing

The final part of the design process was to design the testing methods that would be used for this project. The main method of testing that was implemented was User Testing.

4.8.1 User Testing

To be able to evaluate the game developed throughout the course of this project, a user testing session was designed to be conducted after the Implementation stage. This was intended to help improve the game by having play test the game.

For these tests, a participant information sheet was developed that was intended to inform the user on what it was they were about to take part in, and how they will be assessed during the session. Next a participant consent form was created which confirmed that the participant had read the information sheet and agreed to take part in the test.

Lastly, a Google form was made that would ask the user a set of questions about the game. These questions were:

1. Was the game easy to follow?
2. Explain how well you were able to follow the demo
3. Have you ever player a text adventure before
4. Do you enjoy playing text adventures?
5. Did it feel like you were playing a text adventure when you were playing the game?
6. What was your favourite part of the game and why?
7. What was your least favourite part of the game and why?
8. On a scale of 1 to 10, how much would you want to play the full game if it were released?
9. Anything else you would like to note.

These questions were designed to help evaluate the game to see if it accomplished what was set out in the specification. It was also a way of discerning if there were any changes that could be made that the developer had not thought of or to identify any bugs that negatively impacted the gameplay.

Initially at the start of this project an ethics submission was required to make sure all aspects of this project met the ethical guidelines set by the University of Dundee and any academic bodies involved. All of the previously mention documents were submitted to the University of Dundee ethics committee for evaluation and were given approval as they adhered to the Universities ethical guidelines.

This method of testing was chosen as it is very important part of game design as it allows the developers to gather feedback from other viewpoints to then further develop the game accordingly.

4.8.2 Other Means of Testing

Another way in which the game was tested was throughout the implementation stage, after each new feature was added the program was tested rigorously to make sure the new feature did not cause any bugs. This testing procedure was thorough and tested every element of the game every time a new feature was added.

5 Implementation and Testing

This section is an overview of important aspects of the implementation stage. In this stage the game was fully developed in one sprint.

5.1 Environment Creation

In the specification and design of this report it was decided that the game would consist of four scenes which consisted of slight changes to the terrain and objects. To begin with the terrains for each of the environments had to be created.

5.1.1 Terrain

The first terrain used was a modified version of the terrain created in the third prototype that was described in an earlier section. This terrain was made using Perlin noise to create the realistic terrains, this is also explained in the overview of the prototype. For the purpose of the game, all of the noise values were set so that the terrain had a flat surface and an ascending mountain on two sides of the terrain. This set-up became the base terrain for the game. This terrain will be used in every scene but with slight difference to fit the story and scene design. No changes were needed to the base terrain for the first scene.

For the second scene slight changes were made so that the terrain had the base of a hill that extended to the higher of the two mountainsides. To be able to do this, changes had to be made to the part of the terrain code to where the Perlin Noise algorithm is executed. A loop was added inside the calculate noise function to execute this. This function is where the noise values are taken and then used to define each vertex in a 2D array by looping through both dimensions of the array. For the base hill a set of loops are used to raise the hill. The first loop will run for each row at the range of +10 and -5 of half of the defined row size.

Then the next loop, which is nested within the first, will run for each column that were in the range of +10 and -10 of half of the defined column size. To help visualise this, a 2D array has been created in *Figure 14* below, with the range highlighted in blue. Each vertex in these ranges of values is then changed. For this loop the value is incrementally increased in each iteration of the loop. The result of this is an ascending plane which starts and stops within the range defined.

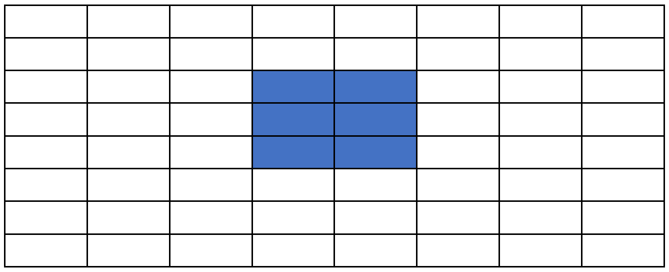


Figure 14: Visualisation of 2D array with points highlighted

Another nested loop which runs from the end of the array to -10 of half the defined size. To help visualise this, a 2D array has been created in *Figure 15* with the range highlighted in blue. Each vertex is changed so it is the same value as the highest value from the previous loop. This effectively linked the top of the ascending plane created in the previous loop and create an elevated path from it and the edge of the terrain. This effectively made the hill which was desired and specified in the specification and is used to create the terrain for the second scene. In *Figure 16* below the output of these loops can be seen.

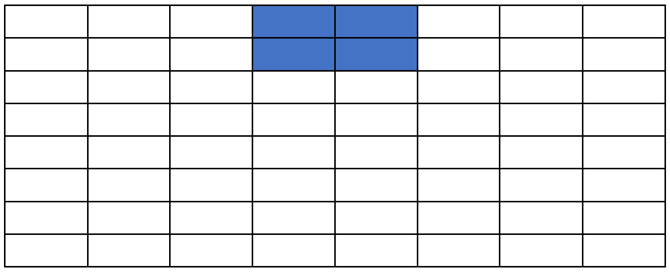


Figure 15: Visualisation of 2D array with points highlighted

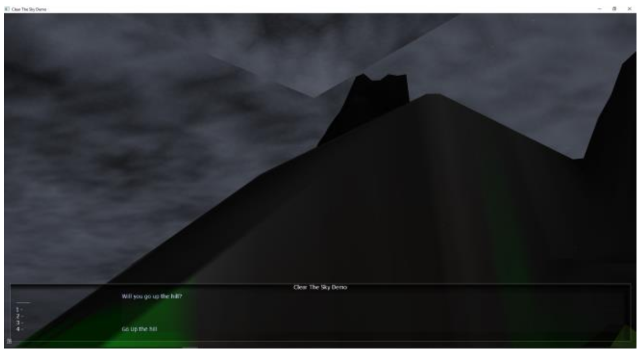


Figure 16: Output of Terrain Code

By taking what was implemented to create the previous terrain, it was easy to extend it to create the desired terrain for the third scene. For the third scene what was specified was a ridge that extended across the terrain in which the player walked along. To create this another loop was added that would run for each row between -5 and +5 of half the size of the defined row size. Then with another nested loop that runs for every column in the rows specified. To help visualise this, a 2D array has been created in *Figure 17* with the range highlighted in blue. Each vertex in this range then has its value changed so that they are at a higher fixed point. This then effectively changes the terrain so that there is a ridge going completely across the terrain. The output of this can be found in *Figure 18* below.

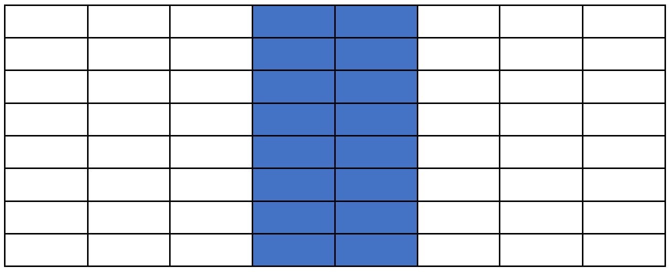


Figure 17: Visualisation of 2D array with points highlighted

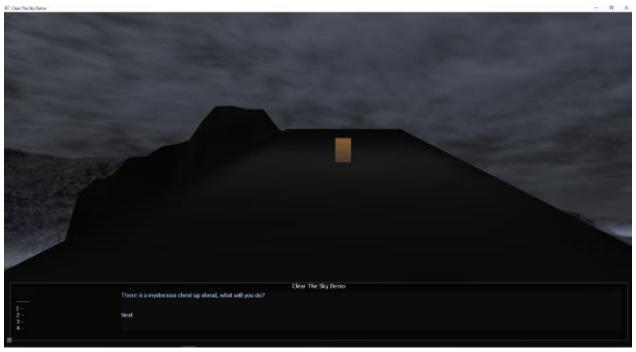


Figure 18: Output of Scene 3 terrain code

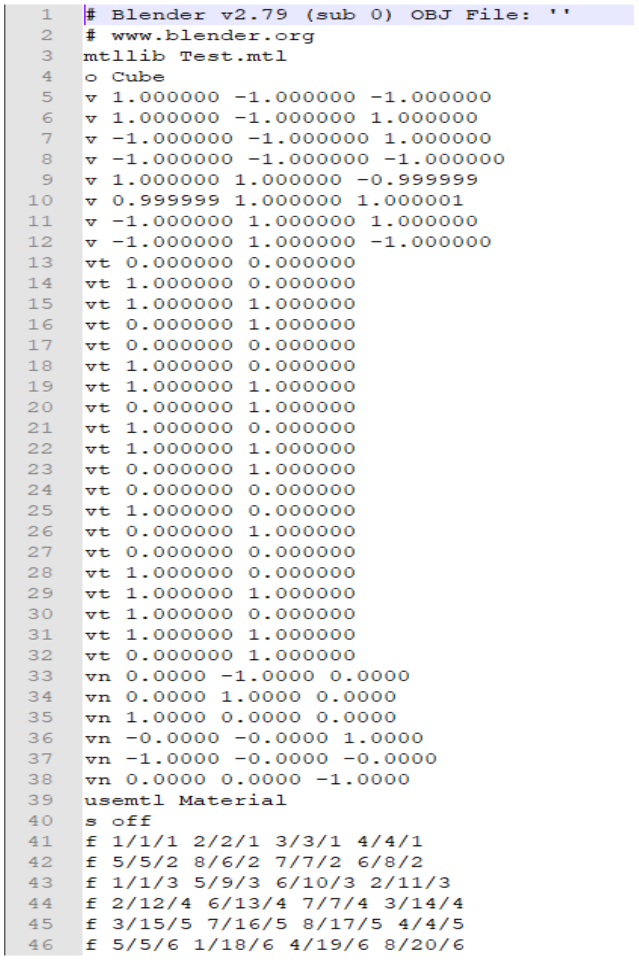
For the last scene the terrain that was originally created for the first scene could be used again as that is what the story and scene design specified.

Whilst creating these terrains a lot of time was spent on getting the ranges for the loops correct. This was especially a problem for scene 2. Scene 2 took a long time to get right because two loops were needed and one of the loops had to be between a certain range and have the right incremental so that the vertical plane ascended in a realistic way. There were many different iterations of the range used for the loop and the most realistic one was chosen.

The ranges were time consuming and difficult to get correct as they were difficult to visualise. This meant that when creating them, the program would have to be compiled and executed to be able to see the terrain visualised, and if it was not correct the values were changed slightly and the whole process was repeated until the correct range was found.

5.1.2 Object Loading

For all of these scenes a few complex objects were needed. To be able to load in these objects an object loader library was needed. To be able to do this, an object file must be read into the program. *Figure 19* is an example of an object file.



```
# Blender v2.79 (sub 0) OBJ File: ''
# www.blender.org
mtllib Test.mtl
o Cube
v 1.000000 -1.000000 -1.000000
v 1.000000 -1.000000 1.000000
v -1.000000 -1.000000 1.000000
v -1.000000 -1.000000 -1.000000
v 1.000000 1.000000 -0.999999
v 0.999999 1.000000 1.000001
v -1.000000 1.000000 1.000000
v -1.000000 1.000000 -1.000000
vt 0.000000 0.000000
vt 1.000000 0.000000
vt 1.000000 1.000000
vt 0.000000 1.000000
vt 0.000000 0.000000
vt 1.000000 0.000000
vt 1.000000 1.000000
vt 0.000000 1.000000
vt 1.000000 0.000000
vt 1.000000 1.000000
vt 0.000000 1.000000
vt 0.000000 0.000000
vt 1.000000 0.000000
vt 0.000000 1.000000
vt 0.000000 0.000000
vt 1.000000 0.000000
vt 1.000000 1.000000
vt 1.000000 0.000000
vt 1.000000 1.000000
vt 0.000000 1.000000
vn 0.0000 -1.0000 0.0000
vn 0.0000 1.0000 0.0000
vn 1.0000 0.0000 0.0000
vn -0.0000 -0.0000 1.0000
vn -1.0000 -0.0000 -0.0000
vn 0.0000 0.0000 -1.0000
usemtl Material
s off
f 1/1/1 2/2/1 3/3/1 4/4/1
f 5/5/2 8/6/2 7/7/2 6/8/2
f 1/1/3 5/9/3 6/10/3 2/11/3
f 2/12/4 6/13/4 7/7/4 3/14/4
f 3/15/5 7/16/5 8/17/5 4/4/5
f 5/5/6 1/18/6 4/19/6 8/20/6
```

Figure 19: Example of Wavefront OBJ file

The format of an object file is as follows: (#) represents a comment like (\\) would mean in C++, (mtllib) references a material file, (v) represents a vertex, (vt) represents a singular vertex's texture coordinate, (vn) represents a singular vertex's normal, (f) represents a face of an object, (usemtl) represents what material is to be used with the object, (s) represents smooth shading, (o) represents named objects.

To be able to load in these files, first attribute structures must be made for each attribute in the file. For the object loader implemented the attribute structures that are created are the vertex's, faces, vertex normal and vertex texture coordinates.

Next, the object file is then parsed into the program. The parser extracts the necessary data and places them in the corresponding attribute structures. Then we process the data by performing indexing on it. Indexing will transfer the parsed data into a new set of variables in a new format so that it is readable for OpenGL. These are then copied into OpenGL vertex buffers and sent down the OpenGL pipeline.

This is an extension of the object loader used in the 3rd prototype. Extensions that were made included the addition of parsing and creating the texture coordinates. This then allowed for the extension to be made so that texture files could be loaded into the program and assigned to the loaded in object.

The implementation of the object loader provided a very important component in the creation of the scenes as it allowed for objects to be loaded in that were created using blender, that meant the use of more complex objects to be used as props in each scene. Most importantly this meant the player had objects to interact with that would allow them to advance in the story.

5.1.3 State Design Pattern implementation

Throughout the creation of the scenes it was difficult as the way in which the scenes were handled sometimes encountered an error where it would generate the wrong terrain or wrong objects. The way in which this is issue was fixed was by using the state software design pattern.

The state design pattern was researched at the start of this project but was not initially used in the design of the project. However, it was the perfect solution for the problems that were being encountered when creating the scenes.

Following this pattern each scene was turned into a state. When the player changes scene, the current state will change so that it is the scene which the player changed to. To accomplish this, a "Scene" variable is used and the value of it is changed so it matches the correct scene. This "Scene" variable is then used in the display() function in Main.cpp to determine what objects and animations are to be drawn and executed for that particular scene. It will also be used to determine what skybox will be used as well.

This variable is most importantly used within the terrain generation code TerrainObj.cpp. It is used to determine which loops are executed when creating the terrain. This guarantee's that the right version of the terrain is generated for the scene which the player is on.

Using this design pattern added to the overall structure of the code and improved it. As well as adding a way to easily discern where each scene is in the code and make it easy to edit it each individual scene throughout development.

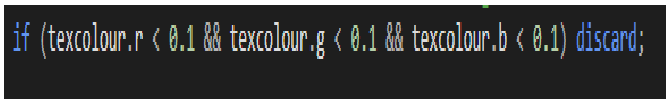
5.1.4 Particle Effects

A finishing touch to the environment creation was the addition of particle effects. The desired effect of the use of particle effects was to make the scene seem alive with dust particles floating around and catching the rays of light breaking through the clouds.

This was already attempted in the 3rd prototype but did not work very well. To be able to create the desired effect the code from the 3rd prototype was adapted.

The particle effect that was used was created using point sprites. Point sprites are OpenGL points that are rendered in the fragment shader with fragment coordinates. A texture is then attached to it, using the fragment coordinates. The unnecessary parts of the texture are then discarded. This is useful as OpenGL points are naturally circular and applying a texture and using this technique can make these points look like different shapes such as stars and snowflakes.

For the game, the texture which was used was a dust particle texture. In *Figure 20*, the texture used is shown before and what it should look like after it has the unnecessary parts of it deleted. For this texture it was any part of it which was black which was deleted. The code from the fragment shader for this is as follows:



```
if (texcolour.r < 0.1 && texcolour.g < 0.1 && texcolour.b < 0.1) discard;
```

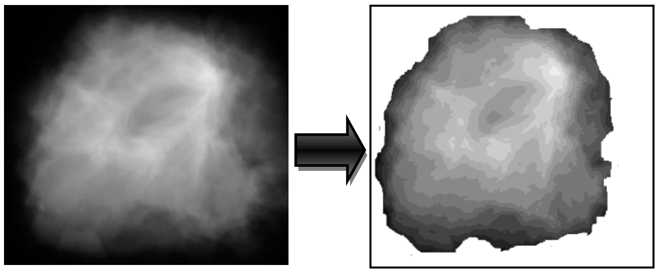


Figure 20: Fragment Shader code and its effect on an image.

These particles are then made using a loop which creates the vertices and colour for each one using a value which is defined when the particle effect is created inside the main program. To move each and individual particle individually a velocity variable was also assigned to each one. All of these variables are given an initial value when they are called. Before animation can begin, the vertex position of each particle has the velocity variable added to it. Then to animate each particle each vertex is assigned a speed from a random range of values. It is also assigned a small value that is to simulate gravity and, in this case, a slight breeze.

For the game the initial start point for the particles was set so that it was on the far side of the scene. The particles also reset once they hit a certain distance from the origin, this was set so it would be the other side of the scene as to save memory and processing power as if they were not reset the particles would just keep moving in the world space straining the computer. Furthermore, a value was added to decrement the Y value of each vertex incrementally to simulate gravity pulling the bits of dust down to the ground. A similar technique was done to the Z value, so the particles were also spread about the scene.

A problem that was encountered was the particle system would make the game run slowly and caused animations to not work properly. To sort this the number of particles was reduced until it no longer impacted the performance but still achieved the desired effect. The particle system is drawn every frame of the game as it is called during the display() function. This makes the particle system constantly up to date and active in every scene.

Overall the particle effect was effectively implemented into the game without negatively impacting the rest of the functionality. The implementation was effective and managed to produce the effect from the specification.

5.1.5 Skybox

To completely immerse the player in the environment a skybox was used to give the environment a 3D panorama that surrounded it. This meant if the player looked around, they could look into the distance in all directions effectively enveloping the player in the environment.

To do this a large cube was made and textures assigned to it to make it look like a landscape. This is done by creating a cube map which is a texture that contains 6 unique 2D textures that form all the sides of a cube. This cubemap is loaded in by loading in 6 individual textures and then assigning them to their corresponding side of the cube.

For the story, multiple skyboxes were needed as the played a part of the game. So, the implementation of the skybox was vital for the environment and was executed well. Examples of these skyboxes that were implimented can be found in *Figure 21* below.

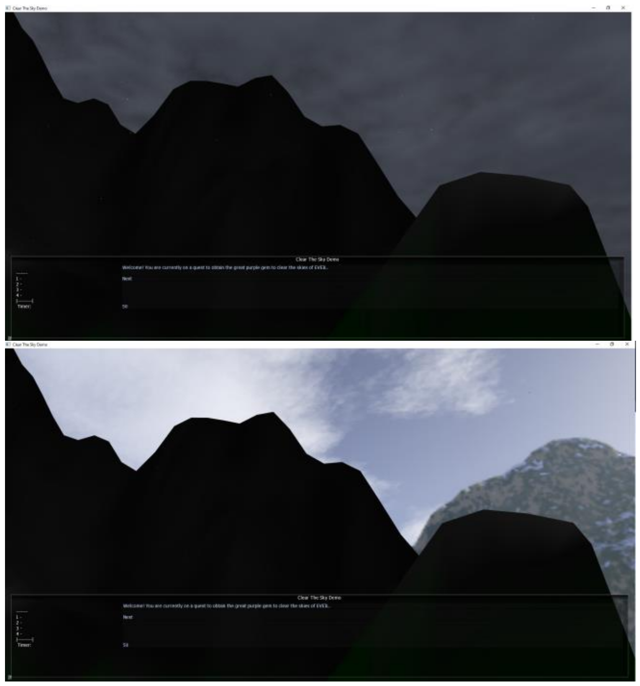

Clear The Sky Demo
Welcome! You are currently on a quest to obtain the great purple gem to clear the skies of EVEIL.
Next
Timer: 58

Figure 21: Examples of Skyboxes Implimented.

5.1.6 Debugging and Evaluation

A lot of time was spent tweaking the terrain and the particle effects, so they were just right and ran smoothly. The time taken to carefully tweak values was worth it as the environment was more immersive and believable overall due to it. It also made the code more efficient and run smoothly.

5.2 Graphical User Interface and Story Implementation

For the user interface the goal was to make the game feel like it was still a text adventure whilst still not taking away from the 3D graphics.

The AntTweakBar library was implemented and used to make the main user interface. This menu was placed in the bottom section of the screen and spanned right across the screen. It was a black opaque colour, so it did not completely obscure the graphics in the background. It contained a line in which the current description of the story and scene was displayed. Underneath this there are up to four lines for all of the current actions the player can take. Then underneath that there is a timer.

Both the GUI and the story had to be implemented concurrently. As AntTweakBar can only display certain variables in specific ways. The best way that to display strings was to use the `TwAddVarRW()` function to assign a variable that contains a string and display it on the bar. The same goes for the players action options. So, the story points will then be stored in a String array. Then when the player chooses the right option it will increment and move forward in the array and the UI will be updated to show the new string in the array.

Once the way to display and play through the story was sorted, the story itself was developed. The story of the game has the player appear in a mysterious land where the skies are always dark and stormy. The players goal is to collect a gem which when placed on a pedestal, will magically clear the sky. This story is designed as a teaser to a wider world and is made to have the player think to make the right choice and be rewarded for that choice.

5.3 Animation

To advance in the scene and story the player must make the right choices. Some of these choices have the player move or interact with different objects which have moving parts. To move the player, their view, and manipulate these objects animations had to be implemented.

To do this the first step taken was get the current time for the start of the animations. This is taken when the corresponding button is pressed. All these animations are handled in the display() function in Main.cpp. Then originally when the button was pressed, the time was set to 0 and the animation played from 0 to whatever number of seconds needed for each specific animation. But this proved to cause problems when running alongside the particle system as well as the animation not running correctly on other systems which were either faster or slower than the development system. This was because the timer system was based on clock ticks of the CPU, and on another system the internal CPU clock either was quicker to tick or took longer to click. To fix this, the solution was to use time delta.

Instead of setting the time to 0 and then animating from there, a variable called "time delta" was created. Then when the correct button to set off an animation was clicked the current time at the time the button was pressed was recorded in a variable instead of being set to 0. Then in the display() function "time delta" is set to be the current time minus the variable recorded when the button was pressed. The position variables of the object or the player is then itself added or subtracted by "time delta" multiplied by the desired value and divided by how long the animation should take. The equation below is the one used to animate components in the program. AV is the position variable that is being changed, TD is time delta, NV is the value the AV is being changed too, and TT is the time taken for the animation.

$$AV=AV+((TD\times NV)/TT);$$

The animations now worked smoothly and succeeded in simulating the movement needed in each scene. They also took a lot of time to create as the values had to be just right to make sure the player was in the right position in relation to the story, this required meticulous testing by free moving the player and have their position printed out to a console after each movement. Then once the right position values were worked out, the value was added to the time delta equation. The time taken variable was then tweaked to make sure the animation took a natural looking amount of time.

5.4 User Testing

After the initial implementation a user testing session was performed. There were 5 user tests completed in this session in total. Each user read the participant information sheet, and once they had done that, they were then asked to sign the participant consent form which means that they have read the information sheet and consent to taking part in the tests. Before the user started to play the game, the author explained a little about the game and told them a bit about how it works and what the controls were. The user then played through the game until the reached the end of the implemented story. They were then directed to a google form that asked them a series of questions related to the game which were detailed in the design section. The author also had a conversation with them about their experience.

5.4.1 Results

Overall the results to the user testing was positive. The results can be found in (appendix results). Overall not many problems were found by the user tests and they provided proof that some components of the specification had been met. The results were analysed, and a few changes were determined to be feasible to make to improve the game.

5.4.1.1 Animation Error

An immediate problem encountered in the user tests was with the animations. When the first animation played if the player clicked the next correct action it would then play the next animation and stop the first this would cause the player to be usually looking at a wall instead of the pedestal that it was meant too. This error happened in most of the tests.

To fix this error the state design pattern was used again. A variable was set up that would define if the program was currently animating or not. Using this new variable, it was set so that when animation was taking place the user controls were disabled so that they could not press any button. This effectively solved the issue that was encountered during these tests.

5.4.1.2 Gameplay User Evaluation

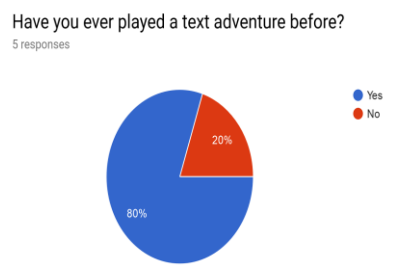


Figure 22: Pie Charts of User Evaluation responses from Google Forms.

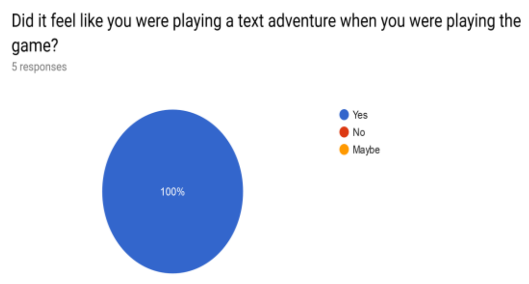


Figure 23: Pie Charts of User Evaluation responses from Google Forms.

Two questions asked in the google form questionnaire were "Have you ever played a text adventure before?" and "Did it feel like you were playing a text adventure when you were playing the game?" The purpose of these questions was to assess if the gameplay met the user's idea of a text adventure as an important part of the specification for this game was that the text adventure elements of the game were apparent to the player. Analysing the data shown above in *Figure 22* and *Figure 23* which are visualizations of all the responses to the questions, there is an overwhelming positive response to the second question. From these results we can ascertain that the game succeeded to fulfil this aspect of the specification.

5.4.1.3 Changes

One of the fields in the user test questionnaire asked users what their least favourite part of the game was. This got a few mixed responses which can be found in *Table 1* below.

Please enter your user ID	What was your least favourite part of the game and why?
1	Did not like that the first two movements were separate
2	Always expecting a timer, so maybe not taking time to fully appreciate all parts of the game.
3	The timed events added too much pressure
4	loading screen hard to follow
6	Text could be clearer and bigger.

Table 1: User test responses to least favourite part of game question.

There was also another question asked which simply asked for any more comments the user may have had. The replies for this can be found below in *Table 2*.

Please enter your user ID	Anything else you would like to note?
1	Add more options!
2	Difficult to switch attention between the timer and the animation. Can the timer only be visible when it is needed? Or if not, put it on top of the animation, so that I don't have to gaze-shift past the text options all the time.
3	Soundtrack was quite good for the setting, decent story
4	no
6	N/A

Table 2: User Test responses to any more comments question.

Analysing this data, it was determined that there were several possible changes that could be made to improve this game.

5.4.1.3.1 Timer Change

The first more important change that was made due to the user tests was to the timer. A couple of the users pointed out that they had difficulty with the timer feature on the main user menu that displayed the story and user action options. The timer only activated once the player chooses the action to press the button which raises the hill and they must then choose the action to run away or the shift in the land would knock them out and they would have to start the game over again.

The problem the users had was the placement of the timer made it hard to notice when the animation before it is playing. Another note was that after the initial timed event, the timer was no longer used but was still displayed. This caused the users to constantly be uncertain about when the timer would go off next. In *Figure 24* below, is the version of the game the user tested. The timer is highlighted by a red outline and is below the player's action options on the GUI and is in quite small text. Due to the size of the image the image can be found in the appendix in full size at (Appendix).

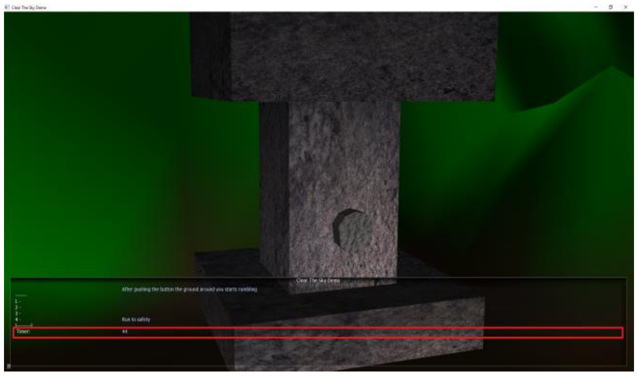

Clear The Sky Demo
After pushing the button the ground around you starts rumbling
1 -
2 -
3 -
4 -
Run to safety
Timer: 44

Figure 24: Screenshot highlighting Timer position

To fix the problems the user had with the timer a couple of changes were made. The first change was to remove the timer from the main GUI and place it in its own GUI bar. This bar is less that half the size and width of the main one and is placed above already existing one in the middle of the screen. Lastly for clarity and to alleviate some of the uncertainty the users experienced, this bar is hidden until the timer is active and disappears when it is no longer active. In *Figure 25* the new timer interface is shown; this can also be found at (appendix).

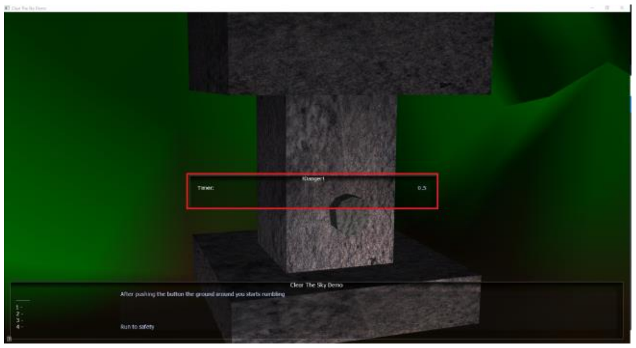

!Danger!
Timer: 0.5
Clear The Sky Demo
After pushing the button the ground around you starts rumbling
1 -
2 -
3 -
4 -
Run to safety

Figure 25: New Timer Interface

5.4.1.3.2 GUI Clarity

Another change that was identified from the testing data was the clarity of the GUI. During the tests some of the testers mentioned verbally that the text on the GUI was a bit small and unclear. Then one of the points a tester made in their questionnaire was the text could be bigger and clearer. AntTweakBar does not have much customization for the text size, but with the options available the text was enlarged and made as clear it could be for the size of the bar and aspect ratio that is supported. Below are *Figure 26* and *Figure 27* displaying the GUI before and after the changes were implemented.

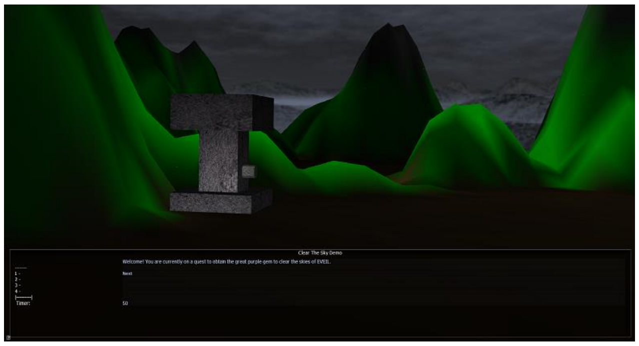

Clear The Sky Demo
Welcome! You are currently on a quest to obtain the great purple gem to clear the skies of EVEIL.
1 -
2 -
3 -
4 -
Next
Timer: 50

Figure 26: Before GUI changes

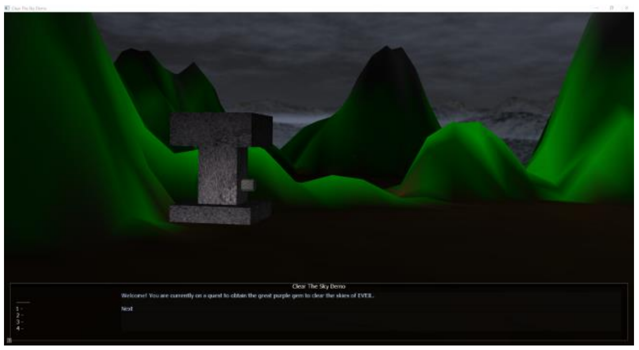

Clear The Sky Demo
Welcome! You are currently on a quest to obtain the great purple gem to clear the skies of EVEIL.
1 -
2 -
3 -
4 -
Next

Figure 27: After GUI changes

5.4.1.3.3 Load Screen Clarity

One comment mentioned that the load screen was a rather unclear and mentioned verbally that they had trouble following it. This was also further shown when observing the testers playing the game, they would all pause for a bit longer on the load screen than expected. This was because they did not see the prompt that told them to press the 1 key to move on to the next screen. Due to the limitations of AntTweakBar there was not much of a change that could be made but the text was enlarged and centred which increases the clarity and more user friendly. This change is illustrated by *Figure 28* and *Figure 29* below.

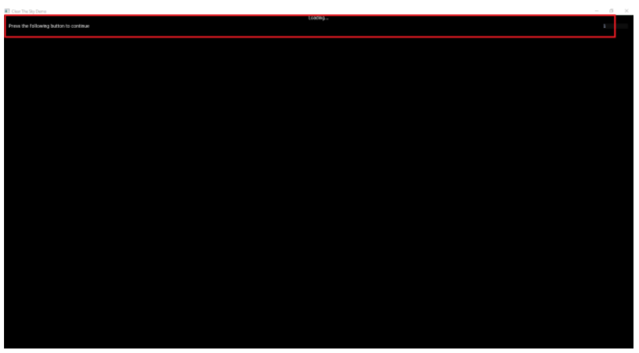

Loading...
Press the following button to continue 1

Figure 28 : Screenshot of Loading Screen before changes

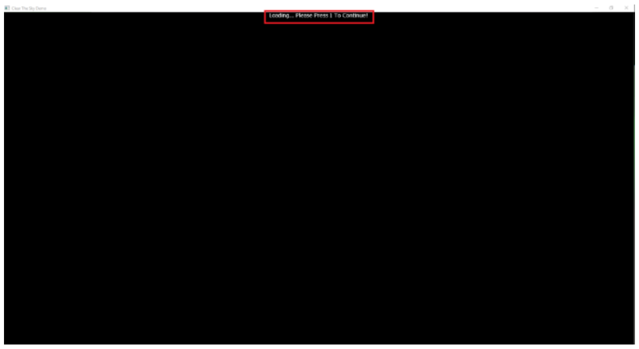


Figure 29 : Screenshot of Loading Screen after changes

5.4.1.3.4 Main Menu

One of the last changes made, was a suggested change by one of the user testers. The tester suggested the addition of a splash screen / main menu. This was implemented due to it being a good idea to polish of the game and make it more repayable. See Figure 30 for an example of the menu.

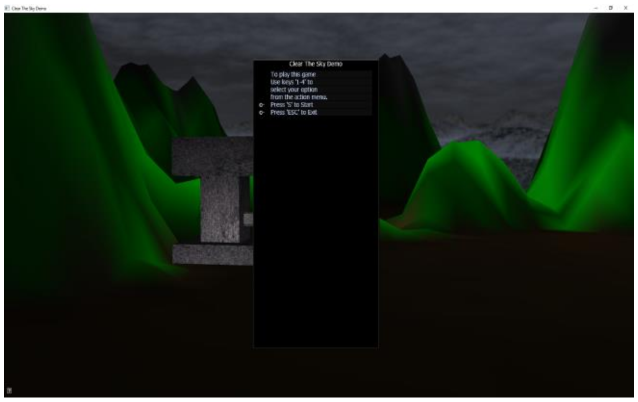


Figure 30: Screenshot showing a main menu implementation

6 Evaluation

Overall the implementation stage was successful. All of the specifications were met, and the game fulfilled the requirements as well. It did take longer than expected to implement all of the necessary features, but this was planned for in the project plan.

6.1 Usability

When it comes to usability a lot of effort went into making sure the game was as playable as possible. User-centred design methods include such methods as rapid prototyping.

First requirements were gathered by researching similar games and the genre for the game as a whole to make sure the requirements for the game adopted some of the features that make it what it is. This overall helped further the project, as well as adding to the usability of the game.

Rapid prototyping was performed to help in research and use different OpenGL methods and techniques. Throughout the implementation stage the game was play tested thoroughly every single time a change was made to the game. These prototyping methods worked well in creating a more usable product as the game was tested repeatedly so most usability issues were found and fixed using these methods.

User tests were conducted at the end of the implementation stage. These user tests helped see how third-parties felt when playing the game. Any usability issues missed during the implementation stage were uncovered using these tests with the help of a questionnaire. These tests were invaluable and helped made some of the changes mentioned in the previous section such as the GUI clarity.

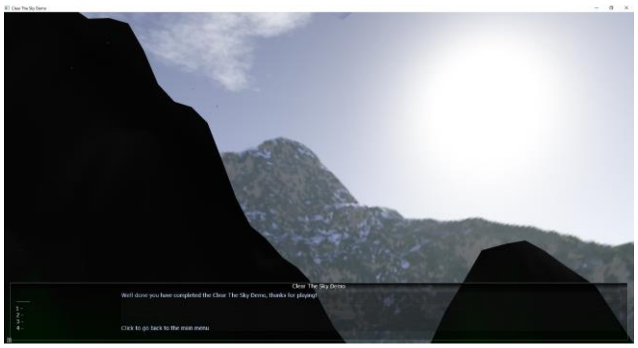
Overall all these methods were effective in creating a user centred computer game.

7 Description of the final product

A text adventure inspired graphic adventure game was successfully created, where in which the player plays through a story where they have multiple options on which actions they take throughout.

The game starts by giving the player a main menu which tells them the controls and how to start the game. When they start the game, they are given a GUI window which gives them the start of a story and gives them options on which actions they can take at that current moment. This GUI is coupled with 3D graphics visualising the scene in a first-person view, in which the story described in the text is taking place.

The player progresses through the game by clicking the correct action options at the right point, this will also allow the player to advance to new scenes and environments throughout the story. The player also must be careful, because if they don't choose the right action at the right time they could be knocked out and have to start over again from the beginning. *Figure 31* below showcases the final scene of the game in its final form.



Well done you have completed the Clear The Sky Demo, thanks for playing!

Click to go back to the main menu

Figure 31: Screenshot of final scene of game

8 Summary and Conclusions

OpenGL was successfully used to develop and test a text adventure inspired graphics adventure game. OpenGL was used effectively to create animated 3D environments. Research was conducted on multiple different environments and prototypes were made to find the most suitable, for the development of the game. After the implementation stage, user tests were conducted that were successful and assisted in making informed tweaks to the game, improving the overall quality and usability of the game by the end of the project. This project also succeeded in advancing the author's knowledge of 3D graphics programming and OpenGL.

8.1 Critical Appraisal

This project was overall successful and produced an end product that the author is satisfied with. Despite this there were several learning points from this project.
Mainly, prioritising was an area in which could definitely have been improved upon in hindsight. Some features of the game could have been more developed if the author had prioritised all features correctly. An example of this was when in the prototyping phase, too much time was spent working on the OpenTK prototype with little progress. In hindsight the first OpenGL prototype should have been started sooner.
Another learning point was the importance of user testing. User testing was a critical component of this project, it was very helpful in identifying undiscovered bugs and quality of life changes. Even though testing sessions were carried out, the author feels more would have necessary to get a more complete user evaluation of the game.

9 Future work

The author believes there are quite a few changes that could be made to the game to increase its quality. One such change would be the implementation of a different GUI library. This is because AntTweakBar was very limited in its functionality and limited other features of the game. One of the features that could be extended with a less limited GUI library would be the implementation of the story. A major limitation was that a string of only a short length can be displayed on the Anttweakbar. This meant that the text being displayed could not be very detailed and descriptive. Another improvement that could have been made was having the story implemented through a text file and a file parser and extended to be longer and more descriptive. The story being longer also means the addition of new 3D environments to fit the new scenes in the extend story. More advanced OpenGL techniques could have been implemented to increase the graphics and lighting in the game, so it was more realistic. These changes would increase the overall quality and extend the playability of the game.

Acknowledgments

The author would like to thank the project supervisor Dr Iain Martin, for his continued help throughout the project. And a special thanks to Hannah McAulay for her constant love and support over the past 2 years.

References

[1] Khronos Group, "OpenGL Overview," 2018. [Online]. Available: "https://www.khronos.org/opengl/" [Accessed 10th Apr. 2019].

[2] Project, T. (2019). Introduction - OpenTK. [online] OpenTK. Available at: https://opentk.net/ [Accessed 22 Apr. 2019].

[3] Project, T. (2019). - OpenTK. [online] OpenTK. Available at: https://opentk.net/faq.html [Accessed 22 Apr. 2019].

[4] Unity. (2019). *Products - Unity*. [online] Available at: https://unity3d.com/unity?_ga=2.205878995.157402306.1555971942-122190935.1555971942 [Accessed 22 Apr. 2019].

[5] Unity. (2019). *Graphics Rendering - Unity*. [online] Available at: https://unity3d.com/unity/features/graphics-rendering [Accessed 22 Apr. 2019].

[6] Unity. (2019). *Unity - Multiplatform - Unity*. [online] Available at: https://unity3d.com/unity/features/multiplatform [Accessed 22 Apr. 2019].

[7] En.wikipedia.org. (2019). *Interactive fiction*. [online] Available at: https://en.wikipedia.org/wiki/Interactive_fiction [Accessed 23 Apr. 2019].

[8] En.wikipedia.org. (2019). *Colossal Cave Adventure*.[online] Available at: https://en.wikipedia.org/wiki/Colossal_Cave_Adventure [Accessed 23 Apr. 2019].

[9] Anderson, T. (2019). The History of Zork - First In A Series. [online] Web.archive.org. Available at: https://web.archive.org/web/20060427000213/http://www.csd.uwo.ca/Infocom/Articles/NZT/zorkhist.html [Accessed 23 Apr. 2019]. Anderson, T. (2019). The History of Zork - First In A Series. [online] Web.archive.org. Available at: https://web.archive.org/web/20060427000213/http://www.csd.uwo.ca/Infocom/Articles/NZT/zorkhist.html [Accessed 23 Apr. 2019].

[10] Lebling, D. and Blank, M. (2019). Zork Trilogy Instruction Manual. Infocom, pp.11-17.

[11] En.wikipedia.org. (2019). *Zork*. [online] Available at: https://en.wikipedia.org/wiki/Zork [Accessed 23 Apr. 2019].

[12] Lexicon A to Z: A Defintive Guide To Gaming Terminology. (1996). *NEXT Generation*, [online] (15), pp.30-36. Available at: https://archive.org/details/nextgen-issue-015/page/n35 [Accessed 23 Apr. 2019].

[13] En.wikipedia.org. (2019). *Myst*. [online] Available at: https://en.wikipedia.org/wiki/Myst [Accessed 23 Apr. 2019].

[14] Breen, C. (1993). A Spectacle Not To Be Myst. *Computer Gaming World*, [online] (113), pp.144-146. Available at: https://web.archive.org/web/20160319021149/http://www.cgwmuseum.org/galleries/index.php?year=1993&pub=2&id=113 [Accessed 23 Apr. 2019].

[15] Cox, D. (2019). Interactive Fiction: Text Adventures. [online] Game Development Envato Tuts+. Available at: https://gamedevelopment.tutsplus.com/articles/interactive-fiction-text-adventures--gamedev-9996 [Accessed 19 Jan. 2019].

[16] Sourcemaking.com. (2019). *Design Patterns and Refactoring*. [online] Available at: https://sourcemaking.com/design_patterns [Accessed 24 Apr. 2019].

[17] Nystrom, B. (2014). *State · Design Patterns Revisited · Game Programming Patterns*. [online] Available at: http://gameprogrammingpatterns.com/state.html [Accessed 24 Apr. 2019].

[18] Nystrom, B. (2014). *Game Loop · Sequencing Patterns · Game Programming Patterns*. [online] Gameprogrammingpatterns.com. Available at: http://gameprogrammingpatterns.com/game-loop.html [Accessed 24 Apr. 2019].

[19] Reed, A., Wardrip-Fruin, N. and Mateas, M. (2014). The Eurekon Design Pattern in Expressive Storygames. Santa Cruz.

[20] Rollings, A. and Adams, E. (2003). *Andrew Rollings and Ernest Adams on game design| Game design*. Indianapolis, Ind.: New Riders, pp.464-465.

Appendices

Appendix A: Ethics

A.1 Ethics submission and response

Appendix B: Design

B.1 Class Diagram

B.2 Scene Design

B.3 Prototype Screenshots

Appendix C: Implementation

C.1 Screenshots

Appendix D: User Testing

D.1 user testing results

D.2 Testing Changes

Appendix E: Project Management

E.1 Gantt Charts

Appendix F: Meeting Minutes

Appendix H: Mid Project Progress Report

Appendix I: Poster

Appendix J: User Manual

Appendix K: Software and source