|  |  |
| --- | --- |
| A picture containing toy, LEGO  Description automatically generated  Everlight Report  P17158642 | Abstract  Everlight is a 2D-platformer game where the players choices affect the outcome of the story.  Github link  Katie Bowker  Computer Games Programming |

Table of Contents

[Introduction 1](#_Toc40656247)

[Project Goals 1](#_Toc40656248)

[Everlight’s Components 2](#_Toc40656249)

[Assets 2](#_Toc40656250)

[Dialogue 2](#_Toc40656251)

[Player Choices 3](#_Toc40656252)

[Level Design 4](#_Toc40656253)

[Bosses and Enemies 4](#_Toc40656254)

[Saving and Loading 5](#_Toc40656255)

[Development Lifecycle 5](#_Toc40656256)

[Assets 5](#_Toc40656257)

[Dialogue 6](#_Toc40656258)

[Level Design 7](#_Toc40656259)

[Player Choices 8](#_Toc40656260)

[Bosses and Enemies 9](#_Toc40656261)

[Saving and Loading 11](#_Toc40656262)

[Documentation 12](#_Toc40656263)

[Testing 13](#_Toc40656264)

[Menu Tests 13](#_Toc40656265)

[Analysis 14](#_Toc40656266)

[Conclusion 16](#_Toc40656267)

[Appendix 17](#_Toc40656268)

[Level Designs 17](#_Toc40656269)

[References 21](#_Toc40656270)

# Introduction

Everlight is a 2D-platformer game where the player’s choices affect the outcomes of the story. The game was developed in Unity alongside the use of the Fungus tool, a free downloadable tool that is used in the Unity engine. This report explores the various components of the game from how these components were planned to be added to how they were implemented into the game. A discussion is also made on how the final game could be improved.

# Project Goals

The overall goal of this project was to create a choice-based game with a diverse choice-based system that not only ensures that the choices the player makes have effects on the game they are playing but that also each time they play the game, they find a different outcome. The game was to be developed with the use of the Unity engine with an important tool, the Fungus tool. This tool allowed the creation of flowcharts to create a choice system where the player can choose how to reply to characters.

The product resulted in four chapters being created. These four chapters work as a ‘Demo’ for Everlight with each chapter including five levels, including the prologue and ending levels, in each chapter with a boss fight at the end of each chapter. Overall, thirty-seven scenes were created, these scenes include the menu and the bad ending. Thirty-seven scenes seemed like a good amount for the demo as it gives the player twelve levels to play with each chapter having three different outcomes to play through.

The main goal of the project was to develop a system where the player can play the game more than once and still be able to experience different things such as changes in dialogue and endings. This was able to be achieved with the use of the Fungus tool and its flowchart system. With this use of this flowchart system, a diverse flowchart path was able to be added to the game.

# Everlight’s Components

Everlight’s components can be split up into six headings. Below is an explanation of what these components are and how they fit into the game’s development

## Assets

In the Literature Review, various asset packs were considered for Everlight. In the end, it was decided that free asset packs created by the user ‘Ansimuz’ on the Unity asset store were to be used. These asset packs were used in the implementation of the game. However, other packs were also used for special effect animations. These additional animations were from the Unity Asset store user, Jean Moreno.

Music and sound effects were also added with the use of asset packs found from the Unity asset store. These, however, were left as the last task to do as the music was not considered as important as dialogue. The music used were from two Unity Asset Store users, ‘marching dream’ and ‘TheoAllen’.

## Dialogue

The dialogue was able to be implemented into the Unity engine with the use of the free tool Fungus. This tool implements flowcharts and various other features into Unity games. These flowcharts were mostly used for dialogue. However, it was found during development that the flowcharts were also able to add audio and animations to the scene. These additional functions, though not related to writing dialogue, were able to add to the dialogue being spoken e.g. certain music playing when a line is spoken or having a certain animation played when the dialogue is finished. It was found that the additions of these functions made the scenes feel more alive.

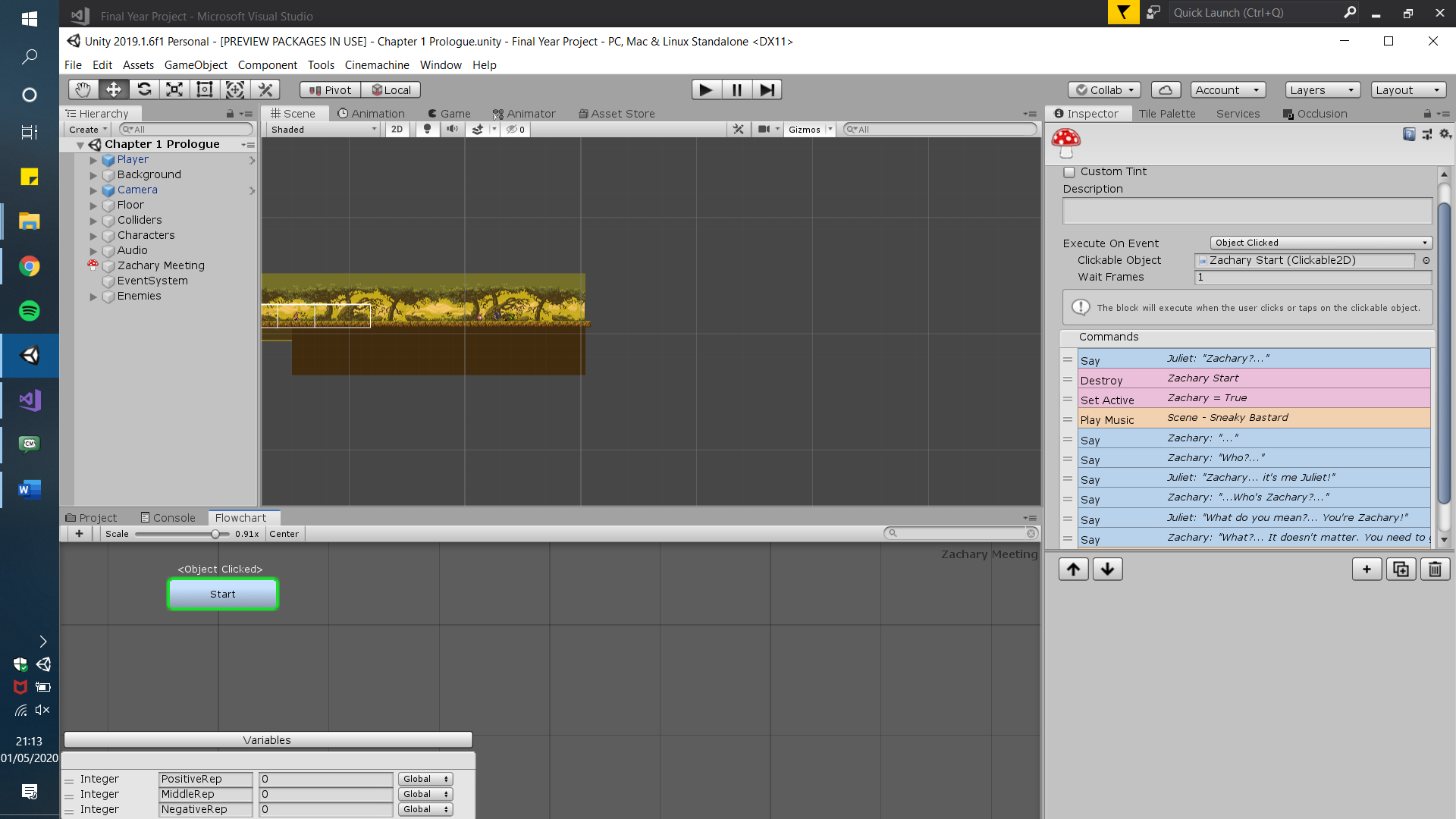


*An example of a Fungus flowchart*

## Player Choices

Player choices were implemented with the use of the Fungus tool. This free tool adds the feature of flowcharts to Unity. When each sprite character was added to the scene, a Character object was then added to the sprite. With this character object, the player can click on the sprite which starts up the dialogue.

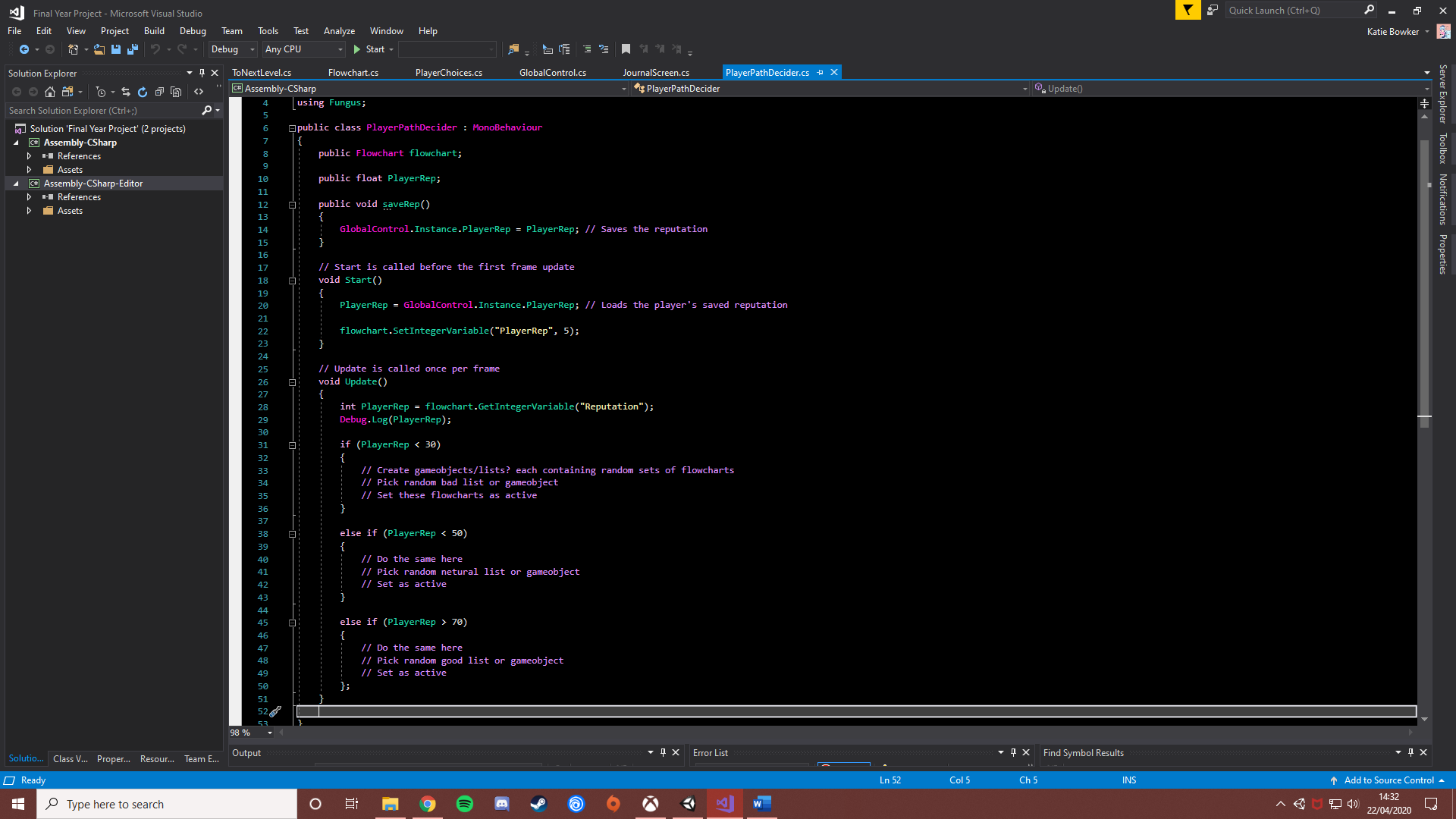
This is all scripted through the flowchart commands feature.



*Above shows how the flowchart is executed is with the Zachary Sprite which is clickable due to the Fungus tool*

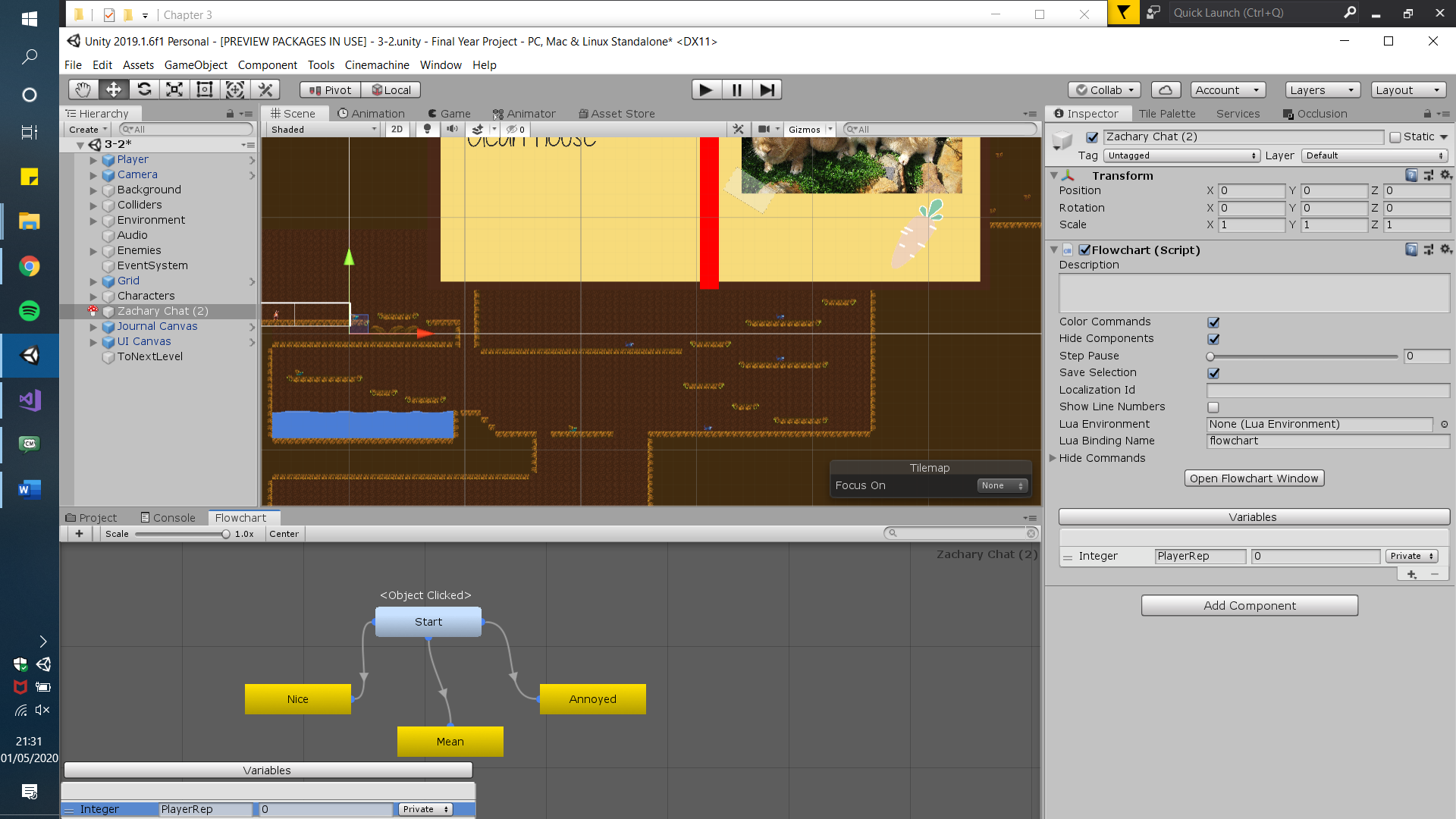
During the literature report, various ways of implementing dialogue to a unity game were explored. In the end, Fungus was chosen due to it being easy to implement and to edit. Most of the ease of implementation was due to the flowchart feature. Each section of the dialogue, the mean, nice and annoyed sections are displayed in the chart. Within each block of the section, commands can be added. These commands are pre-scripted which made setting any game objects active or not active simple compared to having to code the function.

It was planned that as the player reaches a certain point in the game, they will go on a more suited route based on the choices they have made during the game. There were different methods considered for this function. One of these methods was the use of variables with the Fungus tool.



*The Pseudocode for the player’s choices*

The code above is the original plan for how the player choices would have been calculated. An integer was planned to be carried throughout the program. This integer was going to represent the player’s reputation.



*The integer created with the Fungus UI*

How this integer value would have been calculated is determined by what choice has been made. When the player has chosen the nice option, the value would then be multiplied by five. When the mean option is chosen, the integer would have been subtracted by five. Finally, when the ‘annoyed’ option is chosen the integer would have had five added to it. This amount would then used in the code.

Three sets of flowcharts were to be designed for each path. When the code is run, the player’s reputation plays a factor in what set of flowcharts they will be given. When the reputation is checked, the flowcharts under that amount of reputation would then be randomised, meaning the dialogue choices the player was faced with changed each time they played the game. This, however, would not mean that when the player makes the deciding choice, they are set on that one path throughout the entire game. At points, through the dialogue, the player could be faced with a nicer, or a slightly meaner choice. If this option is picked, the player’s path is then changed. This would continue throughout the game to build a large flowchart of events.

However, due to coding difficulties, another system had to be used and can be found in Everlight’s demo. Despite being slightly different in method, the new system shared a similar and simpler idea.

## Level Design

The levels for Everlight were created with the use of tilemaps and Asset Packs that were found from the built-in Unity asset store. Levels were built by using pre-designed ideas on paper. However, the final levels look slightly different than the original designs. As some levels are slightly bigger than they were originally designed.

Levels are set in chapters with three gameplay levels found in each chapter. Each level ends with a dialogue tree except for level 1-1 which is a starter-level to show to the player the type of gameplay they will be faced with.

Endings and prologues to these levels were also created. These were added for several reasons. Reason one being that a ‘bridge’ was needed so that the leap between levels did not feel so sudden. The second reason was for ensuring that the player feels like the choices they have made throughout level made an impact in the game’s story. With an ending, the player can see the impact of their choices when talking to the character who confronts them about their actions.

## Bosses and Enemies

Bosses and enemies were not focused on when creating Everlight as most of the focus was on the dialogue. When the player talks to the character the character’s fate at the end of the boss battle depends on the player’s choices. If the player has been nice to the character, then they will survive the battle, if the player has picked the ‘mean’ options, the character will not survive the battle and if they pick the ‘annoyed’ options the character will be greatly injured. How the other characters respond to the player in the ‘Home’ scenes depends on the state of the last boss the player has just faced.

The boss battle itself has a simple design. The boss sits at the top of the boss area which consists of various platforms and roaming enemies. While the player tries to avoid the roaming enemies the boss at the top of the level fire projectiles towards the player. The player can damage the boss by avoiding all these obstacles and jumping onto the boss. After the player does this five times, the dialogue according to their choices plays out.

## Saving and Loading

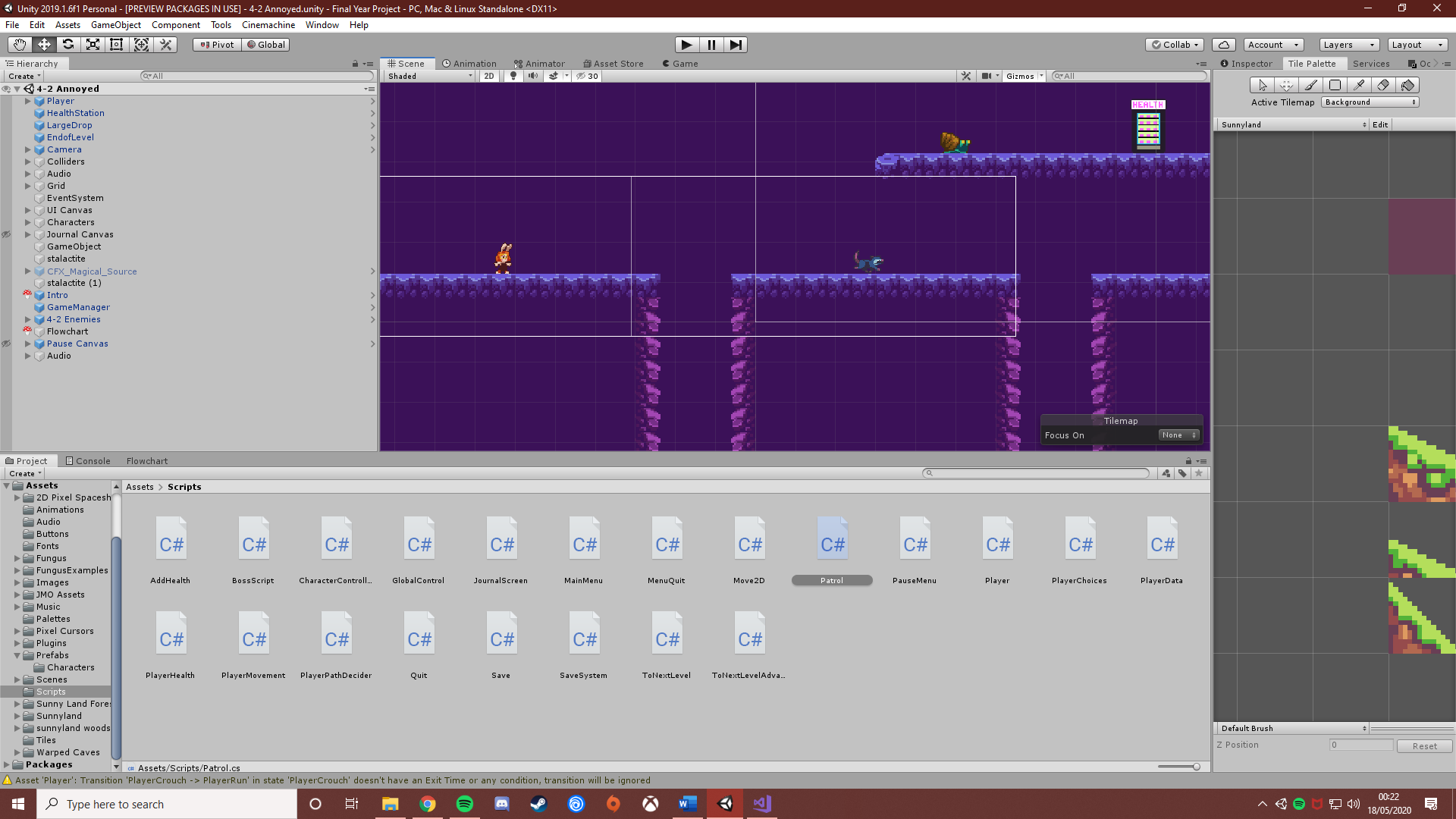
Simple saving and loading features were attempted with the use of several scripts. These scripts were planned to allow the player to save the scene they were currently in. This scene would then be loaded from the main menu screen. However, due to an issue with the code a workaround had to be created as some sort of load feature had to be implemented into the game. This led to the creation of the scene select menu. This allows the player to load in any level except any levels from chapter four and onwards. This was due to the element of choice that is used when loading the levels in chapter four. For example, if you have chosen the nice options in the previous chapter you will go onto the nice versions of chapter four levels and vice versa.

# Development Lifecycle

Here, the development lifecycle of each of the implemented functions is discussed.

## Assets

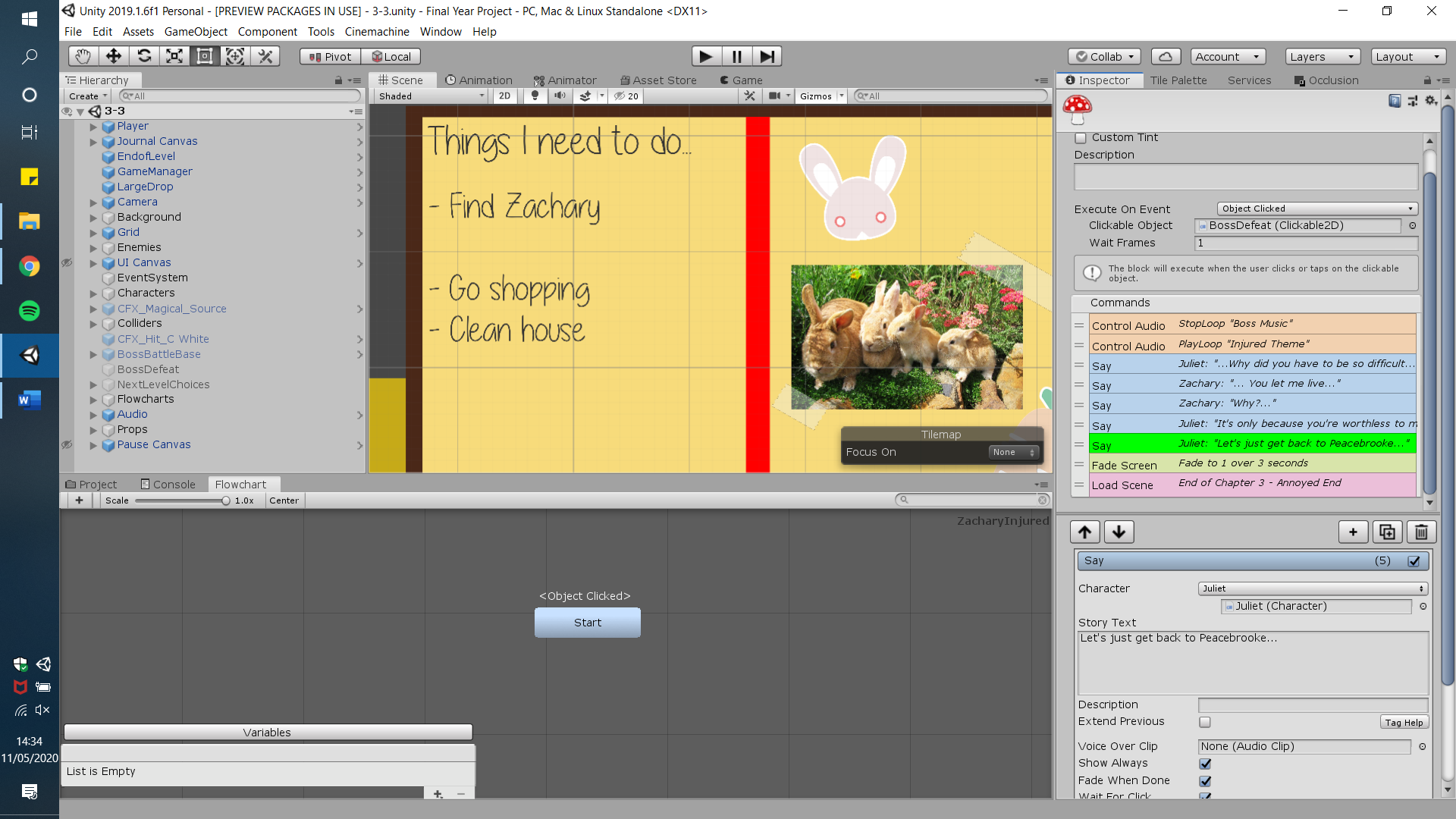
The plan was for the assets to be added as the project was developed. A design for assets did not have to be done as they were all acquired from the Unity asset store. All that needed to be done was to add them to the scenes of the game. With this, creating levels and adding sprites to represent different characters was easy due to the sprites including animations. The only issue came with having to choose sprites. These packs were limited to animal characters and the characters for the game were written before choosing the sprite. So sometimes a character sprite did not seem to fit the character being written. Searches were done for other character sprites on the Unity asset store, but these sprites did not suit the Ansimuz assets, so they stick out. However, this was only a small issue. As it was found that the more time spent adding dialogue and animations to the sprites, they started to fit the characterisation.



*One of Everlight’s levels, all the assets, except the ‘Health Station’ came from the user ‘Ansimuz’*

Despite being happy with the Ansimuz assets in the end more assets did need to be added to the game. These assets were for things like death animations, teleport animations and the bosses fight skill. These additional animations were also downloaded from the Unity asset store. And like the Ansimuz assets, they were easy to implement into Everlight. With the use of Fungus and code, these animations were able to be played at set moments in the game. Making dialogue feel more dynamic and making the end of the boss fight easy to see as a hit animation plays on the player’s final blow.

Adding music and sound effects were the last tasks of this project. Like the sprites and animations, these tracks were downloaded from the Unity asset store. Each chapter has its own set of music with the first two levels having a set track to loop. The third level has a similar looping track as well as boss music that is activated when the boss fight begins. As well as this, each end dialogue has its own ‘theme’, so the music the player is greeted with at the end of the level also depends on the choices they have made. Some background audio was implemented with an audio game object and was also, surprisingly, added with the Fungus tool. Despite only being added to the game for dialogue, the Fungus tool was also found to be useful for the audio. Functions were able to be added to the dialogue flowcharts which could stop or start a music loop.

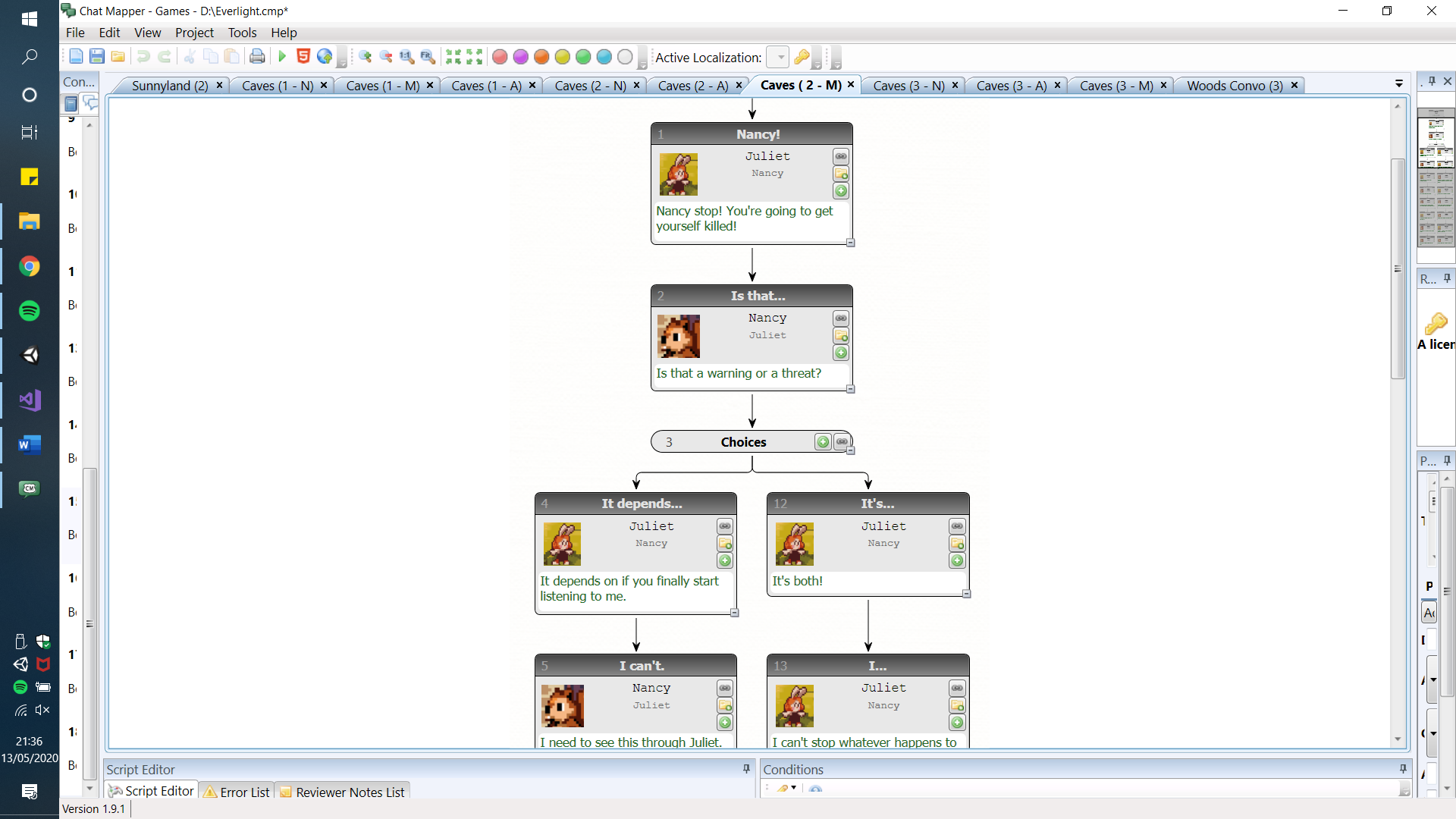
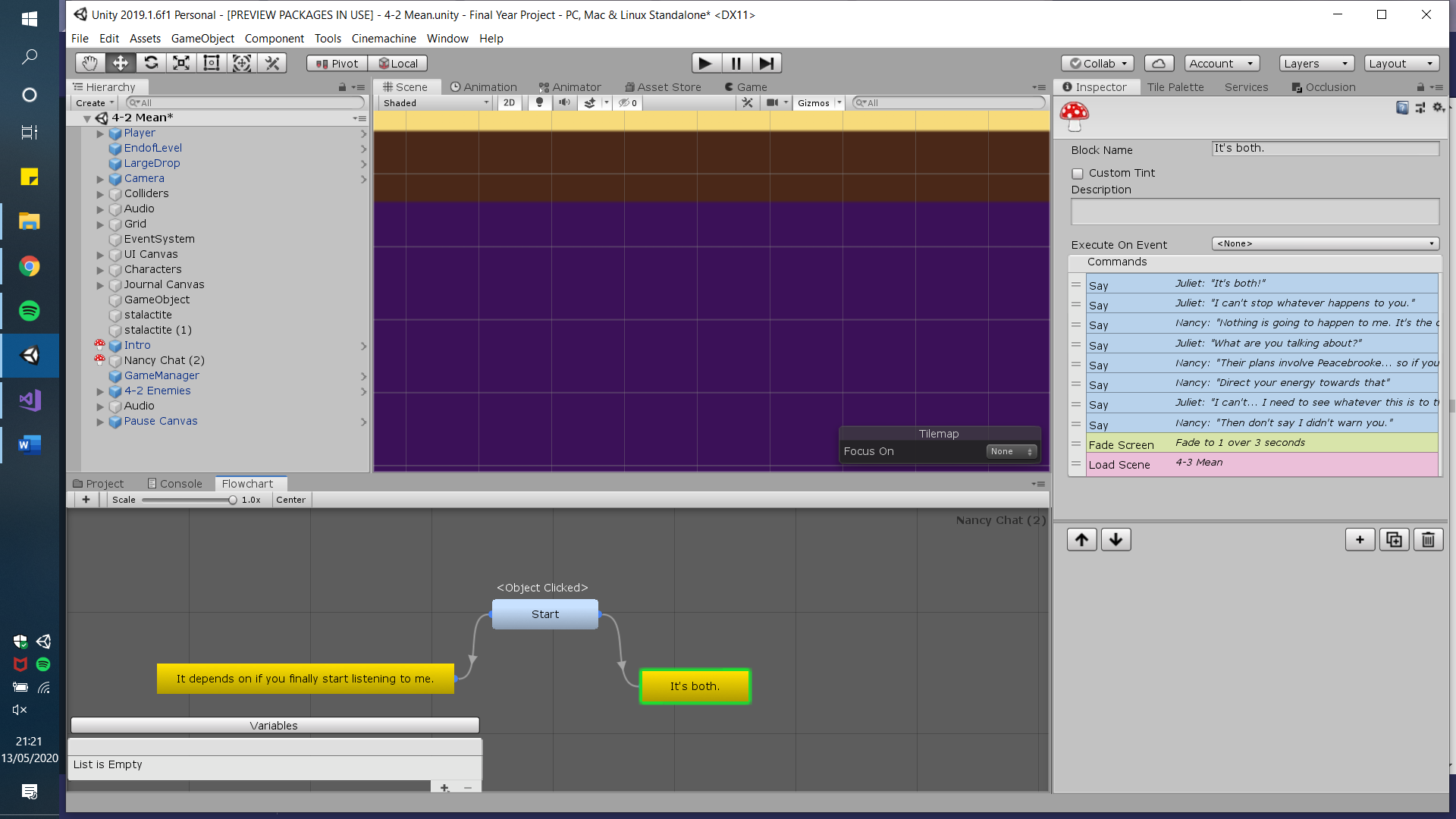


This allowed audio to be played at the exact moment a boss fight was to begin, or when the boss’s defeated dialogue is being spoken.

When each asset was added to a scene a small test would be done to ensure that the asset worked correctly. When the entire scene was completed, a test on the entire scene was then done.

## Dialogue

The plan for the dialogue was for it to be the first thing finished in the project. The dialogue was planned and ended up being written with chat mapper, a piece of software that was mentioned in the literature report. Chat Mapper allows the creation of flowcharts with the addition of characters. This meant character profile for each of the characters in Everlight was able to be created. With these profiles the flowcharts where then able to be created. Each level has one piece of dialogue (excluding level 1-1 which is designed to simply introduce the enemies of the game). This dialogue was written out in flowchart form in Chat Mapper before being added to the game with the use of the Fungus tool. A benefit to using chat mapper was being able to write out the dialogue in flowcharts before adding them to the game. This was simpler as both the dialogue and flowcharts were easier to read through and map out in Chat Mapper compared to when it is added to the game with the Fungus tool.



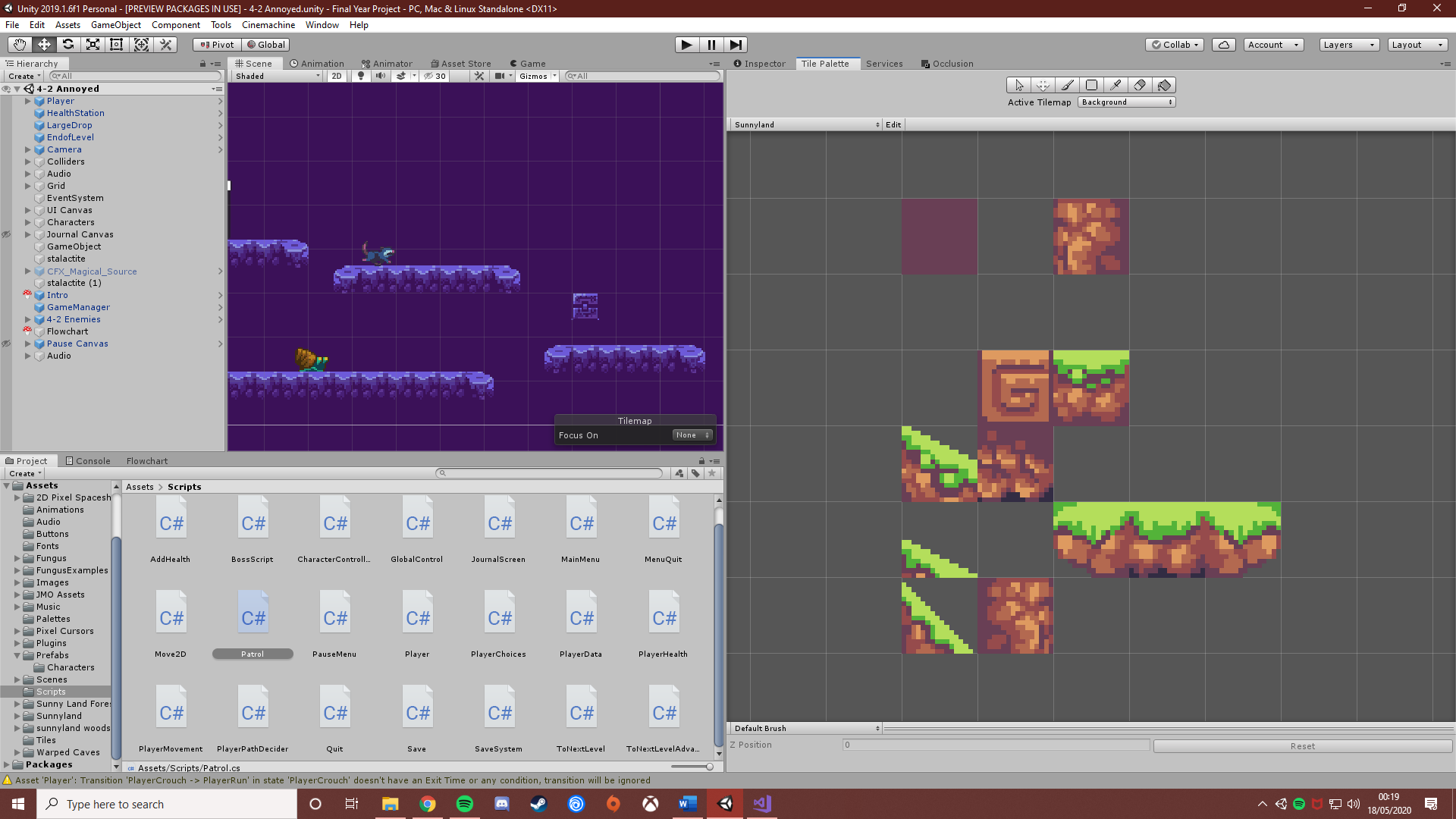
*The layout of the Fungus tool compared to Chat Mappers layout is vastly different*

One change in the plan with the dialogue was the use of one piece of software. This software was called Trelby and was mentioned in the literature review. In the end, this software was not used due to cutscenes not being added in the product. The decision to not include cutscenes was due to the feeling that these scenes would take away the choices from the player. The main idea of the game was that the player has the power to change what happens in the story and cutscenes where the player just watches what happens in the game may take away that element of choice.

Testing had to be performed in two different ways. One way was checking the grammar of the dialogue. Another way was checking that the dialogue ran as expected. Both these tests were performed after the addition of new dialogue.

## Level Design

The original plan was for level designs to be worked on alongside the creation of dialogue. All the levels were first designed on paper, with simple lines showing the general way the level will look and where enemies would be. When putting them in the game, the levels were created with the use of a tilemap and tiles. Using this method of level creation made the process easier as the levels could simply be drawn when being implemented into Unity.



*The ‘Sunnyland’ tile palette, this palette was used to create some of the levels in Everlight, six palettes were able to be created with the ‘Ansimuz’ assets*

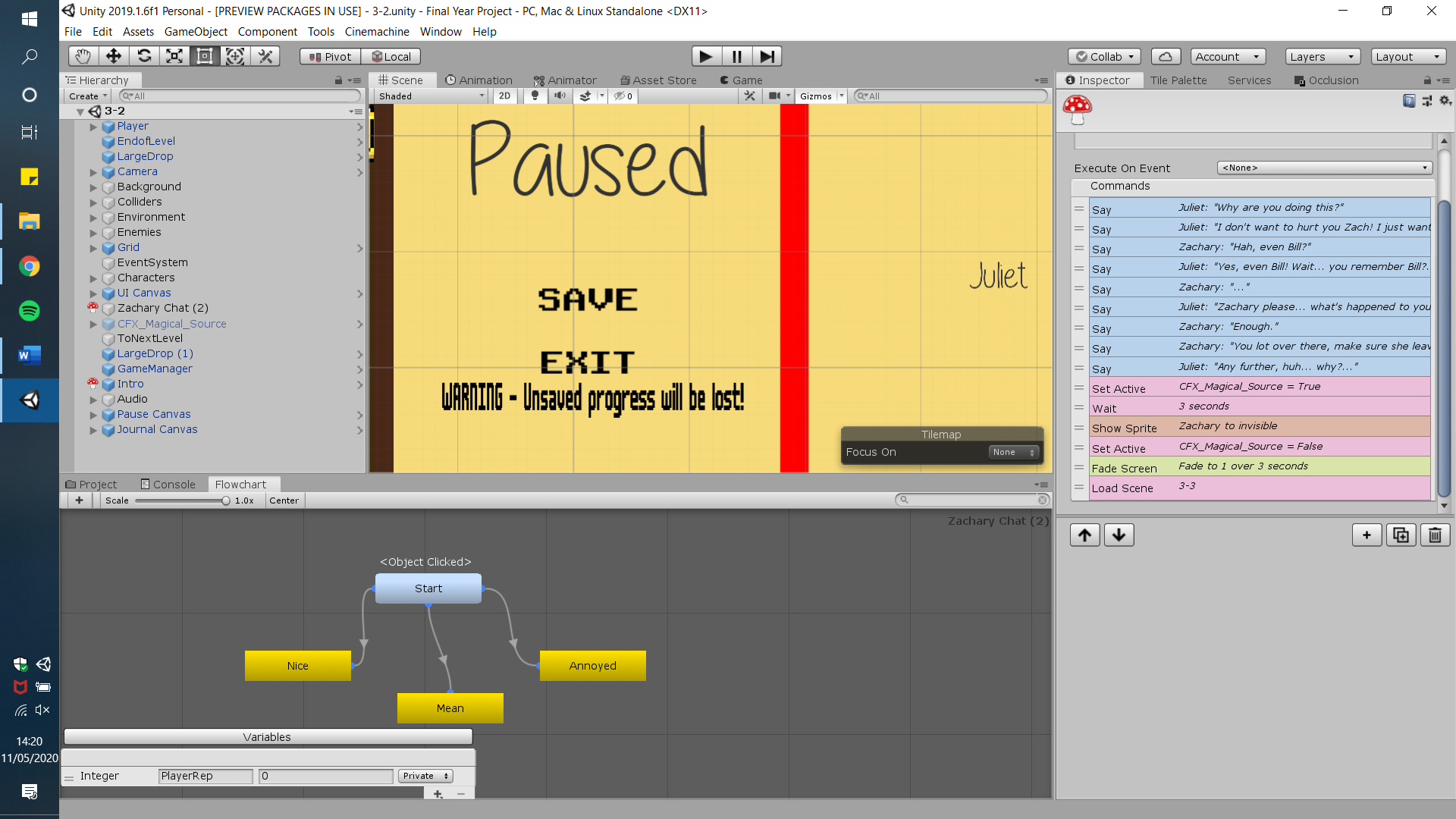
As said before, to make the jump from chapter less sudden a hub level was designed. This hub level is introduced as the first level of the game and sets up the player for the first chapter. However, the original plans of this hub area involved an actual house level where the player could buy skills to upgrade the main character. Eventually, this had to be cut due to time constraints.

## Player Choices

The plan for adding a dialogue system to the game was for it to be developed alongside the implementation of the levels. This decision was made due to the dialogue being added to the end of each level when the dialogue was being added it helped that the dialogue system was also in place so additions would not have to be made later. The actual design of the dialogue system had a few changes during development.

As discussed before, the original idea for the player choices system was for a set of flowcharts to be randomised when the player reaches a certain point in the game. The selected flowcharts would then be set to active to then run in the game. This was all to be done in code and pseudocode was created to create a basic design on how the code was meant to work. However, as development continued more time was spent on developing this code with little results. The main issue with the code was getting the variables from the Fungus function to work alongside the C# code.

However, some type of method or algorithm still had to be developed. The first idea was the use of a diverse flowchart which used both the Fungus features and a simple scene load feature. Instead of different game objects, these various flowcharts were contained within different scenes. The player moves onto these scenes by colliding with a game object called ‘ToNextLevel’. When the player selects a choice on the deciding level the next level that loads when they collide with the ‘ToNextLevel’ game object depends on the players choice. This method was used with some slight changes. The game objects did not need to be used to the functionality of the Fungus tool. A load scene function could be used within the dialogue trees. So, when the player’s conversation finishes, the next level is then loaded.



This was done easily as the load scene function as scenes can be applied to the function by simply typing the name of the next scene. This made the last chapter of the demo easy to implement despite there being three versions of the same level. This function also made it easy to have the player switch routes. In the last chapter on the ‘annoyed’ route, the player is given the chance to reply to the character in a nicer tone than they have previously replied to them in. If the player selects this option, the player is then moved onto the nicer route due to this choice. By adding choices like this, it is ensured that the player is given varied options and that the number of different routes the player can take is increased.

Another benefit to using this system is with the extra ending which is known as the ‘Bad Ending’, the player is presented with this ending a few times and if it’s chosen a simple dialogue box pops up which tells the player what that ending entails for the character. After this scene plays the menu is loaded up allowing the player to play the game again to try and get a different ending. Again, this was all easily achieved with the use of the Fungus tool.

Despite this new system removing the random-like nature of the game, this method still opens the number of choices the player is presented with and the number of different paths the player can go down.

After all, this testing still had to be done on the system. Like the previous functions, tests were done each time the system was implemented in the flowcharts. When the system was fully-added to the demo an overall test was done on all the levels to ensure that each level links to the correct level.

## Bosses and Enemies

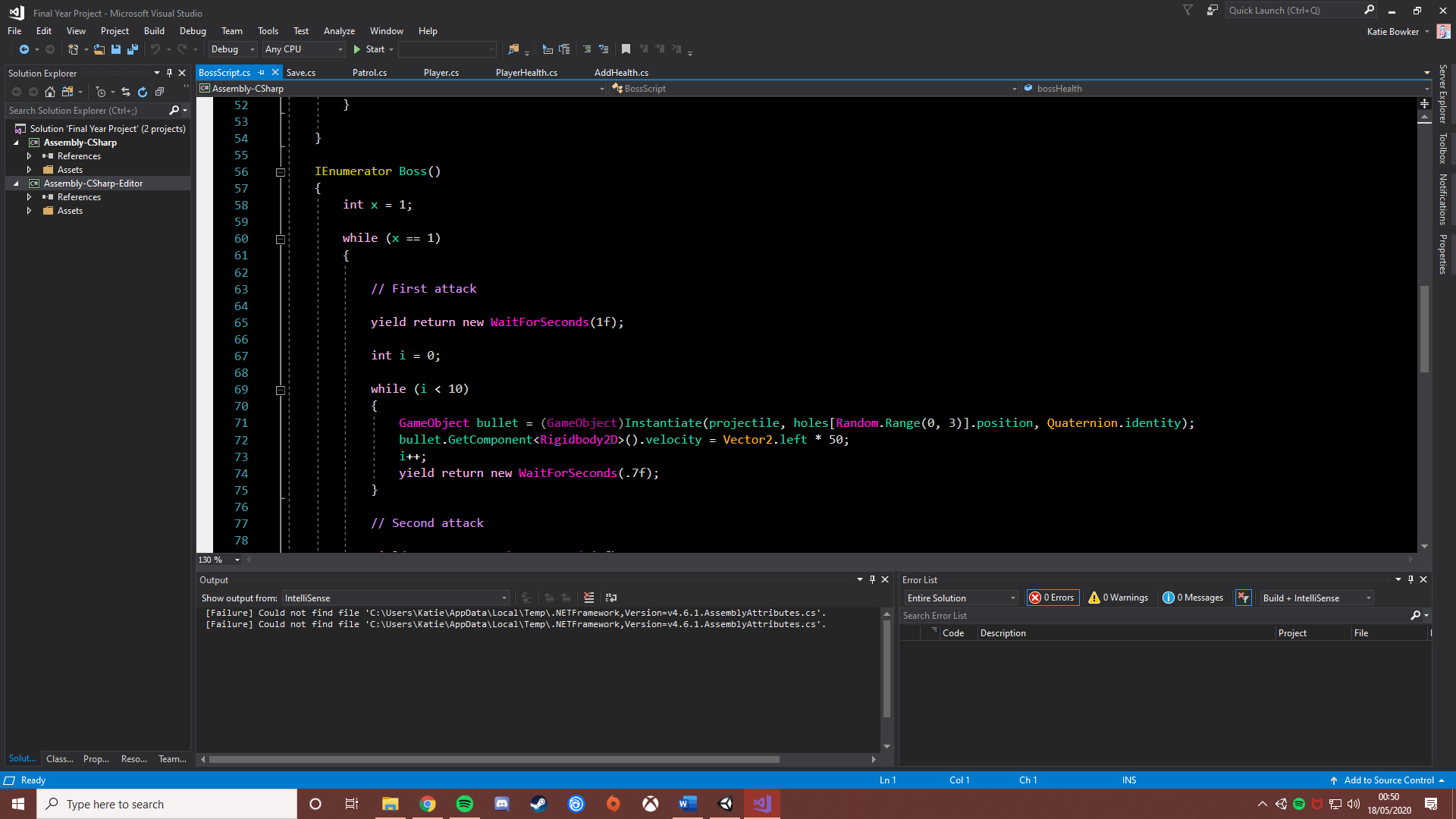
The plan for bosses and enemies was for them to be added at separate times. Enemies were added first during the creation of levels due to their simplicity. Enemies have a simple patrol script implemented into them. A sensor checks if the sprite has reached the end of a box collider if they have then they turn and start walking in the opposite direction. After this, a player health script was added. This script is responsible for health being taken of the player when they collide with an enemy. The player cannot fight the enemies they can only avoid them. Other dangers that the player may face could be the level itself from water to dangerous falls into sharp rocks. These types of obstacles will instantly kill the player, resulting in them having to restart the level.



*The patrol script added to every single enemy in Everlight*

Tests only had to be done on each enemy once as each one was turned into a prefab. However, each time a level was finished each enemy was double-checked to ensure they worked correctly.

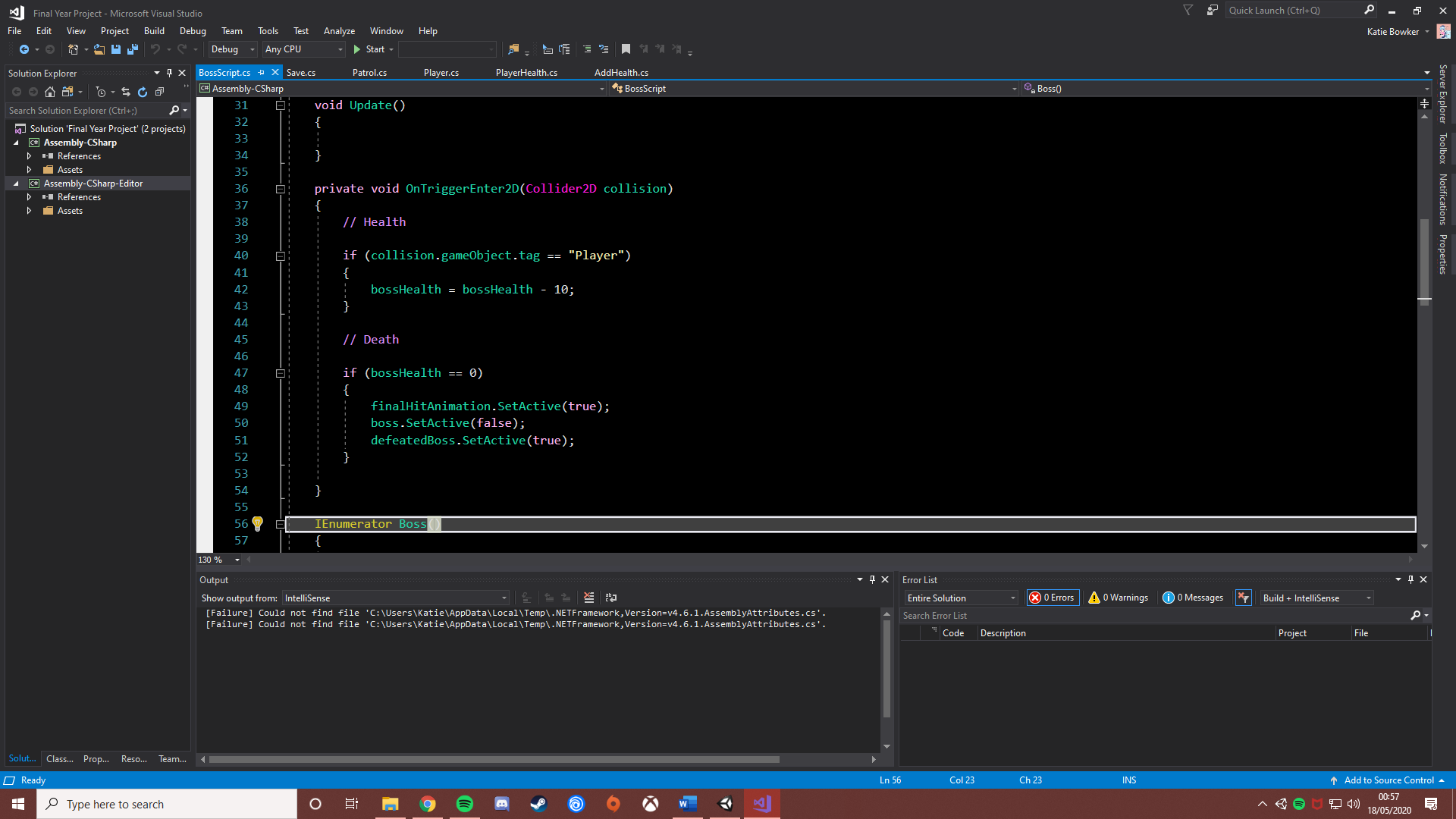
Bosses were added in after enemies due to being more complex to add. A plan for them was not created due to being implemented with the help of a tutorial video. Bosses were able to be implemented with a script known as ‘bossScript’ from a tutorial from a YouTube user known as ’Wabble – Unity Tutorials’.



*The code for the bosses’ first attack, all the attacks use a similar type of code*

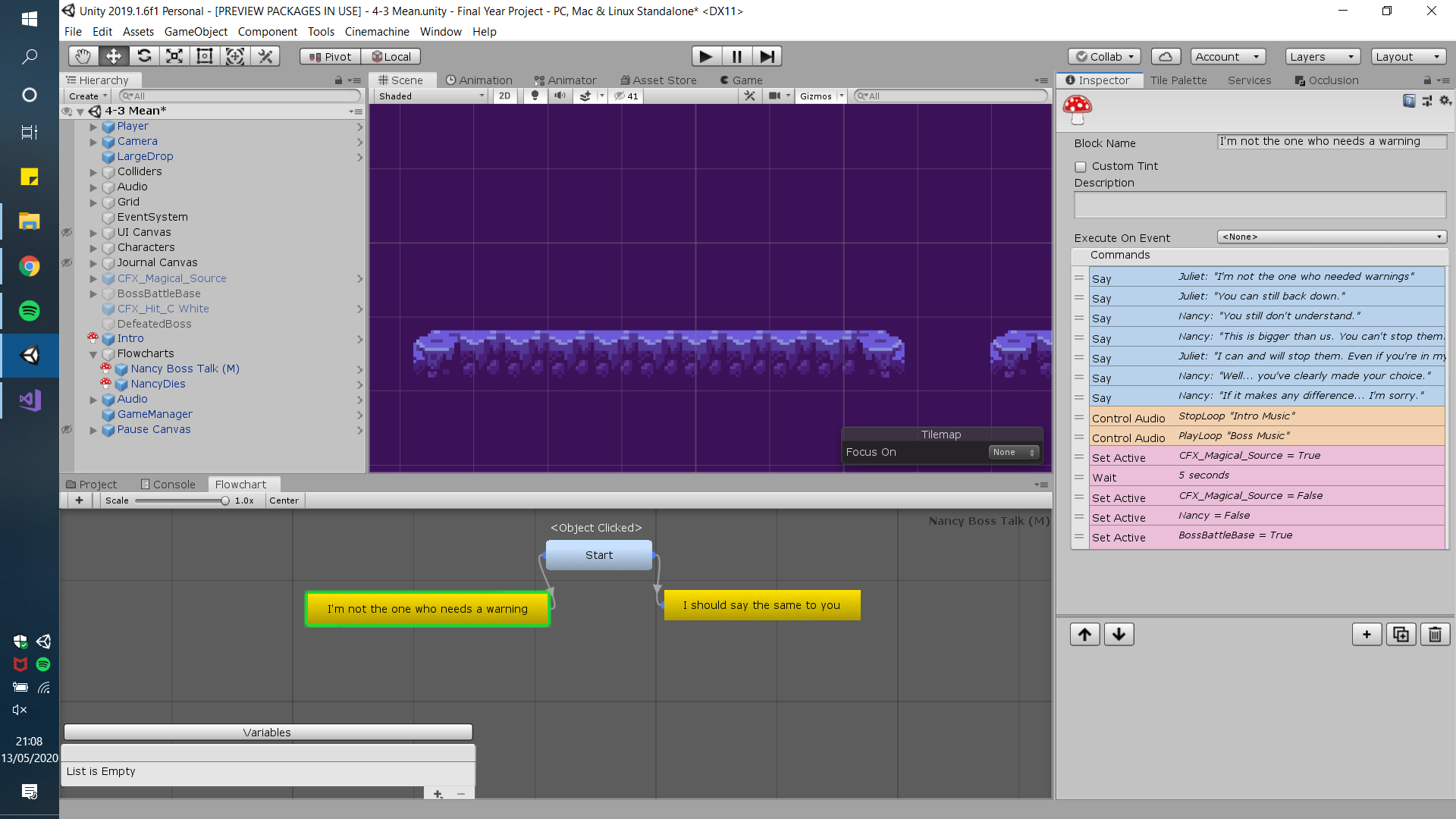
This script added ‘holes’ to the boss. These holes would shoot out fireballs from random holes that have been set on the boss. These holes were just simple game objects positioned around the boss. With this, the boss can fire out ten fireballs above, left and right in a loop until the player defeats them. Each fireball takes away one of the player’s hearts.

As well as giving the ability to shoot fireballs, various other variables were able to be added to the boss with this script. Some of these variables were the boss ending animation and the defeated boss game object. These objects are enabled when the boss’s health is found at zero. The boss’s health was coded with the same method used for the player’s health. When the boss’s health reaches zero the boss deactivates, and the defeated boss game object activates which starts the boss’s dialogue.



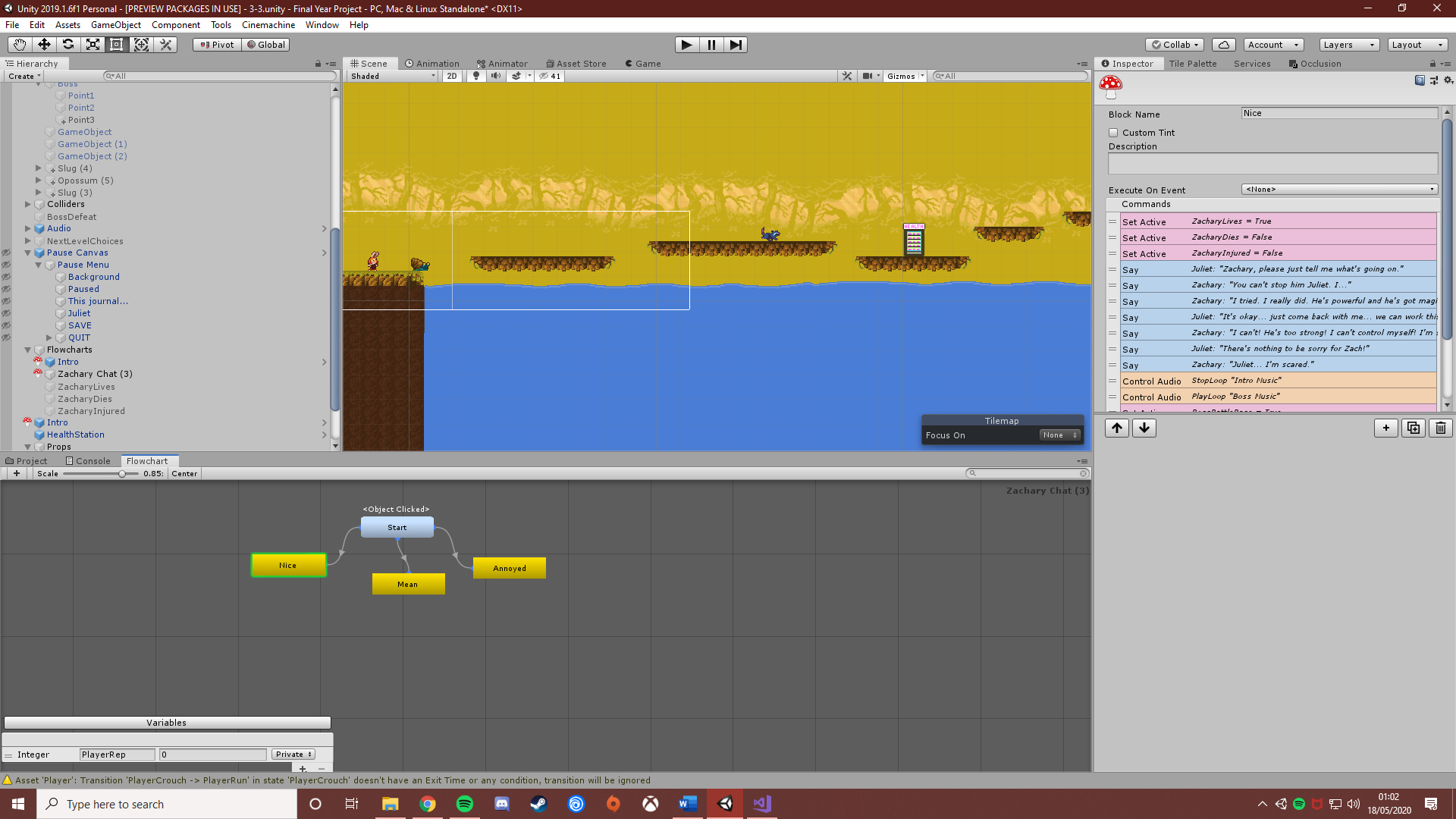
*The code that allows the player to defeat the boss*

Like the way audio was implemented, the Fungus tool was found to be quite useful when adding the boss fights. The first way in which Fungus was able to be used for boss fights was for starting the actual fight. With Fungus’ set active functions the boss battle object that contained all the enemies and the boss itself was able to be set as active when the player is finished talking to the boss.



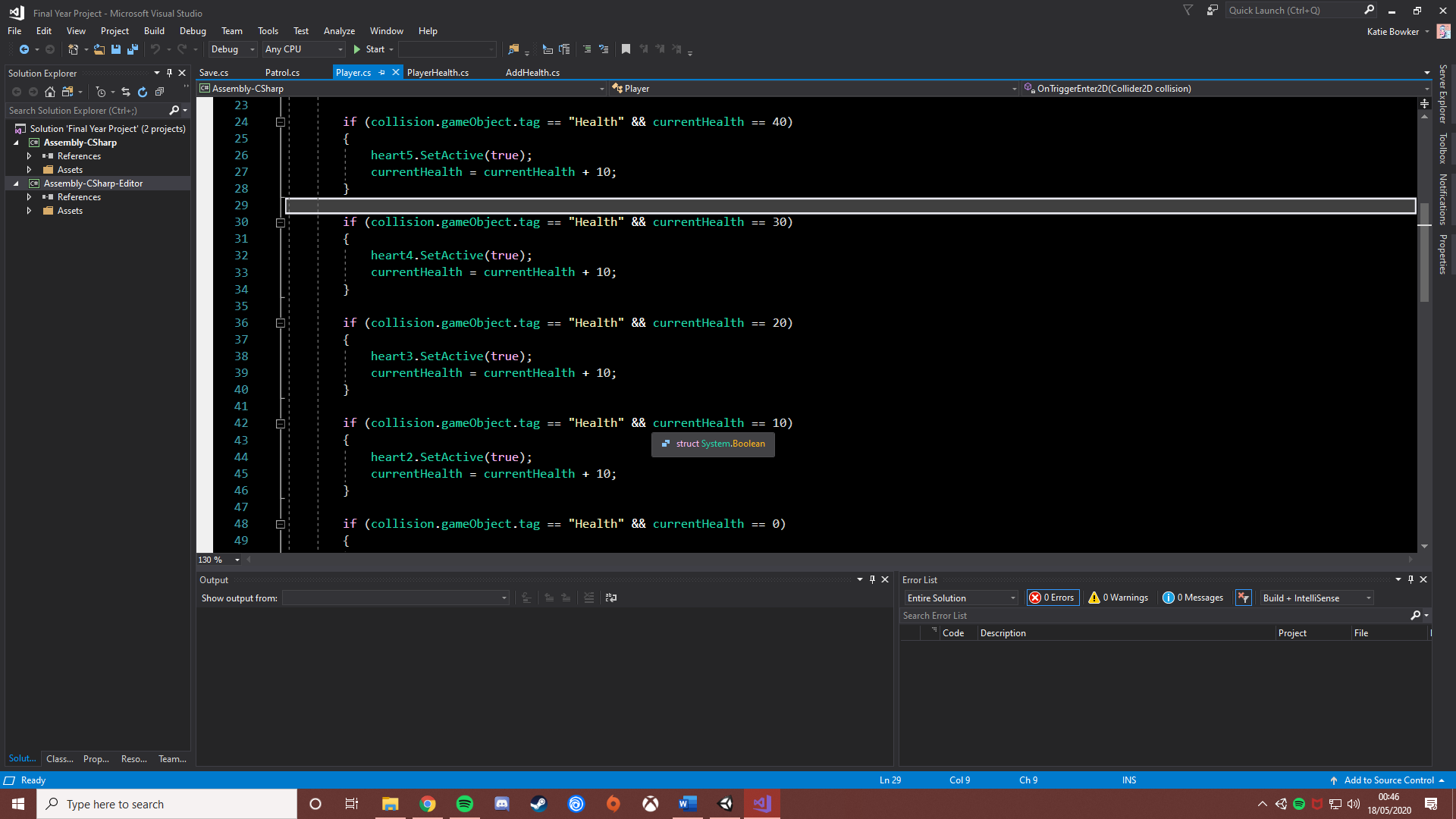
*The Fungus functions used to set the boss as active*

As well as setting the boss as active Fungus was also able to have the player’s choices affect the outcome of the boss fight. In chapters one to three, the outcome of the boss fight depends on the choices the player has made in the dialogue leading up to the fight. However, in chapter four this slightly changes. When the player makes their decision at the end of chapter three the route, they are now on is then decided. When the player reaches the end of chapter four the decisions, they make during the pre-boss dialogue does not affect the outcome of the fight, it is determined by the route they are on.



*The functions used to set the required flowchart as active*

A last note on bosses is the addition of ‘Health Stations’ when testing it was found that even though the boss levels were not extremely challenging, it was difficult to defeat the boss if the player has low hearts. This led to the implementation of healing stations. The ‘Healing Station’ itself was an original design done on the website ‘make pixel art’. With the image created all that needed to be done was the addition of code for it.



*The code that heals the player*

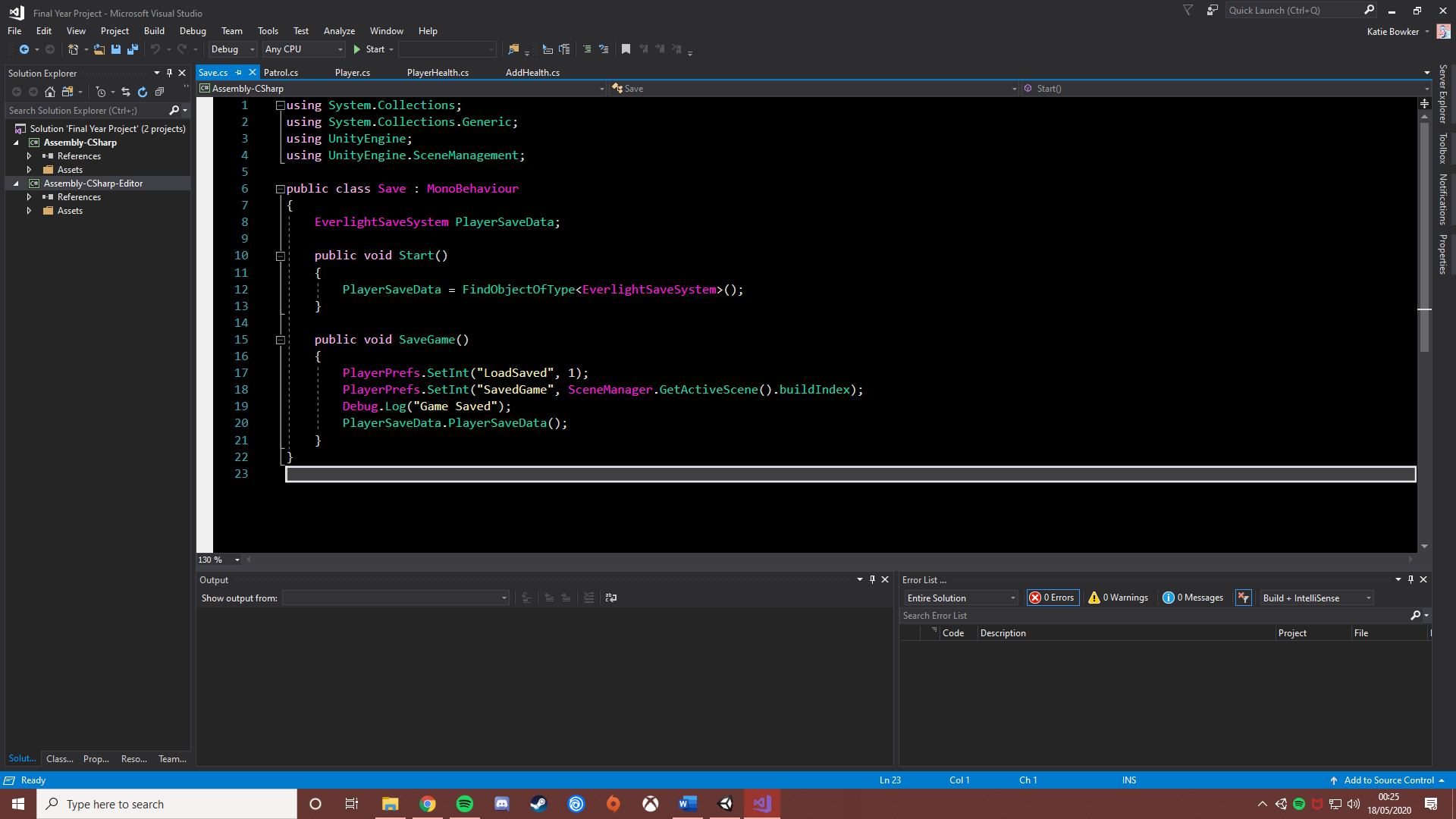
This was done in the player script and it was coded in the same way the player’s health is taken away when interacting with an enemy object. With this implemented the player was able to heal and find the bosses easier to defeat.

Again, tests were done after each boss was added. The second set of tests were not needed to be done on the bosses as they were the last features to be added to the levels.

## Saving and Loading

The plan for the saving and loading feature was for it to be one of the last features implemented. This was because the game would be able to still meet its requirements with or without a saving a loading function.

Like the bosses, a plan was not designed for the feature as a tutorial was used to implement the save and load function. Scripts were added to try and add a save and load function. The scripts used for this function were the scripts ‘Save’ and ‘Main Menu’. Both these scripts contained the functions required to save the game when in a level and to load the game when in the main menu. This was attempted with the use of Player Prefabs. When the player hits the ‘Save’ button, an integer named ‘LoadSaved’ is set. Another integer named ‘SavedGame’ is then set by getting the currently active scene. With this data the main menu is then able to have a load function, this load function uses the ‘LoadSaved’ integer, if it has been set, then the ‘SavedGame’ integer is used. This allows the player to load the previously saved scene.



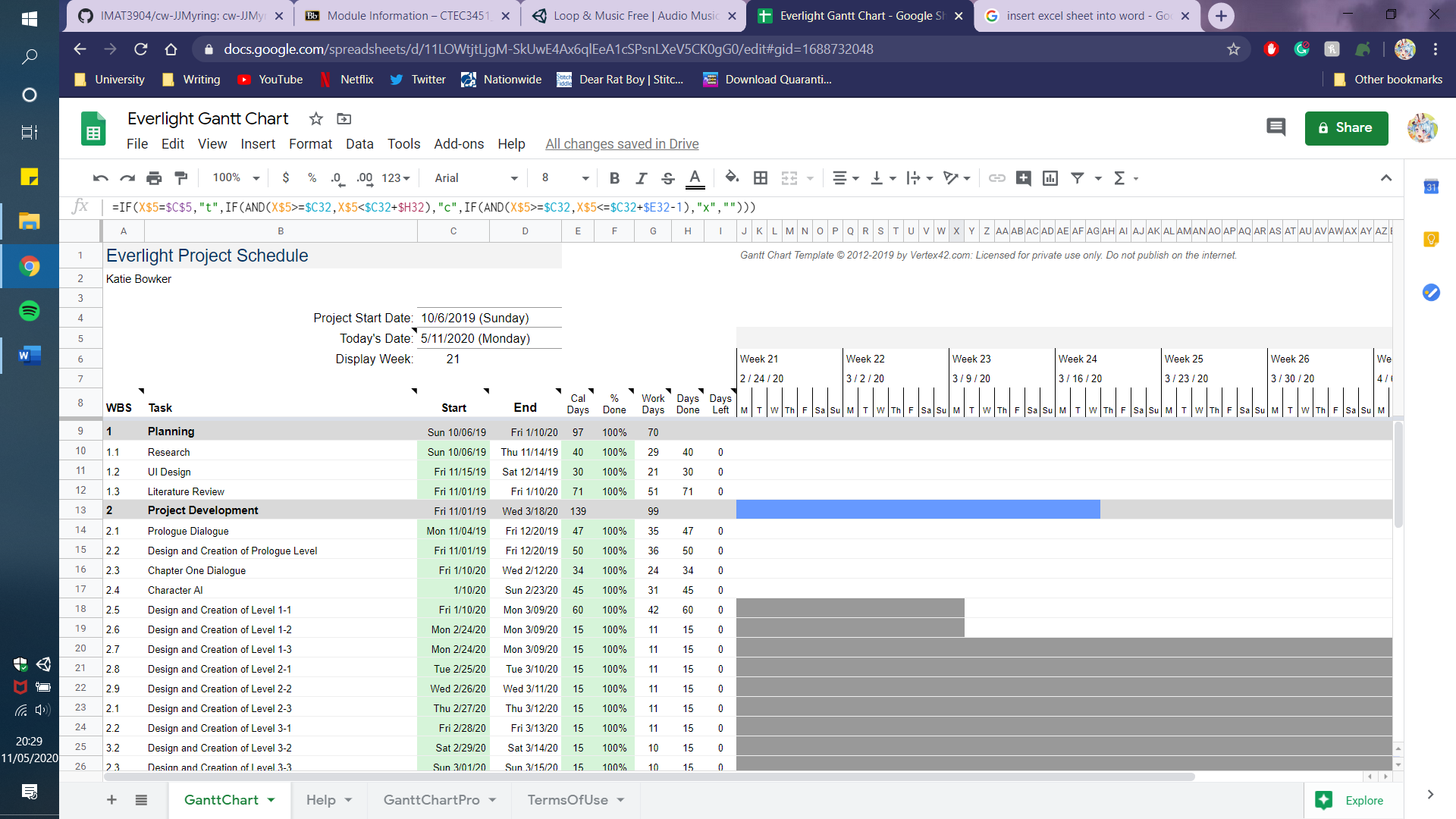
*The attempted ‘Save’ code*

However, an error did occur where the player was able to save the game but when the player tried to load the game from the main menu, the scene loads but the player cannot move. This error was found when testing the function after implementation.

Due to this error, this system had to be left out and replaced with a ‘Scene Select’ button on the menu. This button would allow the player to load any chapter they want excluding chapter four.

# Documentation

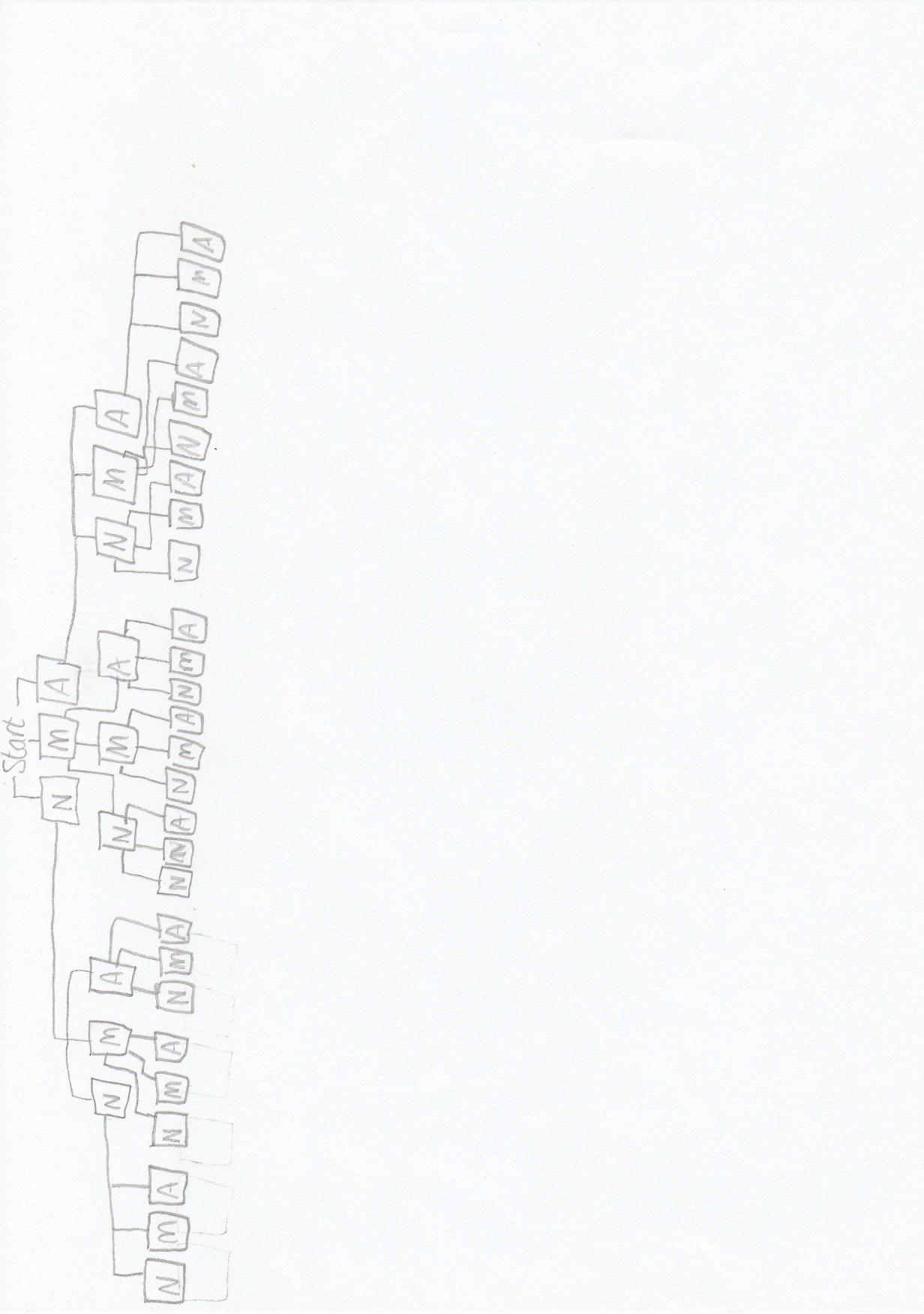
When creating a project like this documentation was needed to be kept. This documentation would show the progress of the project day by day. This project schedule was kept in the form of a Gantt chart.



*Everlight’s Gantt Chart*

Despite being created, there were times that the Gantt chart was forgotten. This was mainly due to the use of daily to-do lists, these lists would be used and checked off, but the Gantt Chart was forgotten but still updated every week.

Another piece of documentation was the level and flowchart plans. Both these plans were sketched out on paper. Below shows a sketch of the flowchart structure, the level sketches can be found in the appendix.



*A sketch of the basic idea of the flowchart structure*

Having a sketch of the levels being implemented made creating levels much easier and faster. This was because all that needed to be done when creating the levels was to copy the sketches done on paper into Unity with the use of tilemaps. Enemies were represented by crosses on the level sketches and again, were easier to add to levels because they just had to be put in the place of the crosses.

Finally, a sketch was also created for the flowchart structure. This was sketched out to get a clear view of how each flowchart linked into each other. Only a basic sketch of the beginning of the flow chart could be sketched as the chart got more complicated to sketch.

# Testing

Testing was done in the form of white-box tests. Test cases were re-done when new features were added to scenes. This was done to make sure that the addition of new functions had not broken other functions. Each scene in Everlight was tested with individual test cases. However, the levels that have different versions e.g. the endings and the levels in chapter four, are tested in the same table but are coloured so that it easy to tell what test cases are for what version of that level. The menu testing has been included within this report to show the way the tests were carried out; the rest of the tests can be found in the testing document found within the project files.

## Menu Tests

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Case | What is being tested | User input | What should happen | What did happen | Comments/Changes made |
| 001 | Play button | The user clicks the button | The scene changes to ‘Prologue’ | The scene changes | N/A |
| 002 | Scene select button | The user clicks the button | A menu pops up showing the scenes that can be selected | The menu displays | N/A |
| 003 | Chapter one button | The user clicks the button | The scene changes to ‘Prologue’ | The scene changes | N/A |
| 004 | Chapter two button | The user clicks the button | The scene changes to ‘Interval 1’ | The scene changes | N/A |
| 005 | Chapter three button | The user clicks the button | The scene changes to ‘Interval 2’ | The scene changes | N/A |
| 006 | Scene select back button | The user clicks the button | The player is returned to the main menu | The main menu is shown | N/A |
| 007 | Credits button | The user clicks the button | The credits menu is shown | The credits menu is shown | N/A |
| 008 | Credits back button | The user clicks the button | The player is returned to the main menu | The main menu is shown | N/A |
| 009 | Quit button | The user clicks the button | The game closes | The game closes | N/A |

# Analysis

In the analysis of the entire project, various things were implemented well and other things that could be changed or added to improve the project.

Several functions were planned to be in Everlight but were not able to be implemented into the end demo. One of these functions was the ability to buy skills. These skills were planned to be simple gameplay additions e.g. double jump or special powers like projectile shooting. There were several reasons why this feature was unable to be added into the demo. The main reason was due to timing. When the decision was made to work on this function it was found that there was not enough time to add all the functions required for this feature. This function would require a skill shop that would be able to be used at the start of every level, the bought skills to be displayed on the journal screen and these skills to be added to the gameplay. When this task was started to be undertaken it was found that there was simply not enough time to implement this feature properly, so it was left out.

A similar function that could be added is are coins and magic that were planned to be spread over the levels for the player to collect. The feature was added to the UI, but the functionality was never implemented. The reasons for this are linked to the skills function. The gold and magic the player collected were purely for the skills function where the player can buy and apply skills to the character. Because this skills function was never added, it felt redundant to add the magic and coins to the game. However, the areas that the gold and magic were planned to be added to were filled with the ‘Health Stations’ so that these areas were not left bare.

Another function that was planned to be added was the Journal. A type of journal was added into Everlight, but some features were missing. The original plan for the journal was for buttons to be placed at the side of the book. These buttons would display different interfaces such as a task list, friends list, skills list and settings. Only the tasks page was able to be implemented in the demo. The reasons the other pages in the journal were not able to be implemented were due to the same reasons was not implemented. Time was an issue when it became time to implement the journal pages. This is because the journal feature was left as one of the last tasks to implement. This choice was made because the journal seemed less important compared to the other tasks such as getting the choice system to work and the creation of dialogue.

Although a workaround was created, saving and loading is another function was unable to be added. The reason for this was due to difficulties in coding. If done again, it would be preferable to have this feature implemented as there were other ways the feature could be added. One of these ways is with the Unity Asset store. This was not used as the saving and loading asset packs needed were required to be bought. If done again, one of these asset packs would be bought to ensure that the saving and loading function is implemented successfully.

In retrospect one area which other improvements could be made was adaptability. Due to outside issues, free time that would have been created due to exams was no longer available due to exams changing to coursework. This coursework ended up taking the free time that was planned for the project. With better adaptability, both these tasks could have been balanced more effectively. This also brings on another area that could have been improved which was timekeeping skills. One main issue was knowing the most important tasks that needed to be done first. This lack of schedule made it hard to focus on one task and led to different tasks being done at the same time. Despite getting the most important features of Everlight added such as boss fights and different routes. Additional features such as a skill shop or a working save and loading feature were unable to be added. A Gantt chart was created for the project but due to forgetfulness, the chart was forgotten and when features were planned to be added to Everlight, the tasks were simply just done without adding them to the chart.

Next time, to improve the timekeeping of the project a Gantt chart, although used slightly this time, could be used throughout the project development. Another way timekeeping and managing different tasks could be improved next time is by setting different Sprints throughout the project. These sprints would have set tasks that need to be completed by a set date. This may help with sorting through tasks and deciding which ones need to be done by when.

A final area that could be improved is the prologue levels. These levels work for the requirements needed but some changes can be made. However, a lot does not need to be done to improve these levels. The only improvement that could be made is the addition of further dialogue. This addition of further dialogue would also require additional levels. There would have to be three prologue levels per chapter due. This is because three versions of the level would have to be created, one version for the ‘nice’ path, one for the ‘mean’ path and one for the ‘annoyed’ path. This was done with the fourth chapter as separate prologues needed to be used for the player to get sent down a path suited to the choices they have made. However, the dialogue is not different in these separate prologues they are just there for functionality.

Despite all this there are still some areas in the project that did work out and, in some cases, came out better than was expected.

One of these areas was the choice system. Due to finding difficulties in coding a coded algorithm could not be created. However, a system was still able to be implemented with the Fungus tool. With this tool, a function was able to be implemented so that at the end of the dialogue the required level is then loaded. This simple feature made the idea of having several scenes for one chapter more manageable. With this, Everlight can have a branching flowchart with several routes the player can go down. Despite there only being four chapters, the player is given a large amount of playtime due to these different routes.

A final note on player choices is with the endings of the chapters and the ‘bad end’ the player can come across. The ending scenes the player sees depends on the choices they have made. An example of this is if the player has been ‘nice’ throughout the chapter then they will get a ‘nice’ ending. Alongside this, each type of ending has its very own theme. The result is dialogue that feels cinematic and a reason for the player to try and achieve different endings as they all include different pieces of dialogue.

Another area that came out better than expected were the levels themselves. This is largely due to the assets chosen. The original plan for Everlight was for the game to have a cutesy look and this plan was achieved with the use of Ansimuz’s assets. These assets are bright and colourful and are a perfect fit for the game.

A final area in development that was handled quite well was the changes made to functions when issues with coding came up. Despite some functions not being able to be implemented as planned due to difficulties in coding these functions were still able to be added despite being different than they were planned to be.

# Conclusion

This project aimed to create a story-based platformer game. With these requirements a 2D-Platformer game was developed where the players' choices not only affect the route, they go down but how boss fights end.

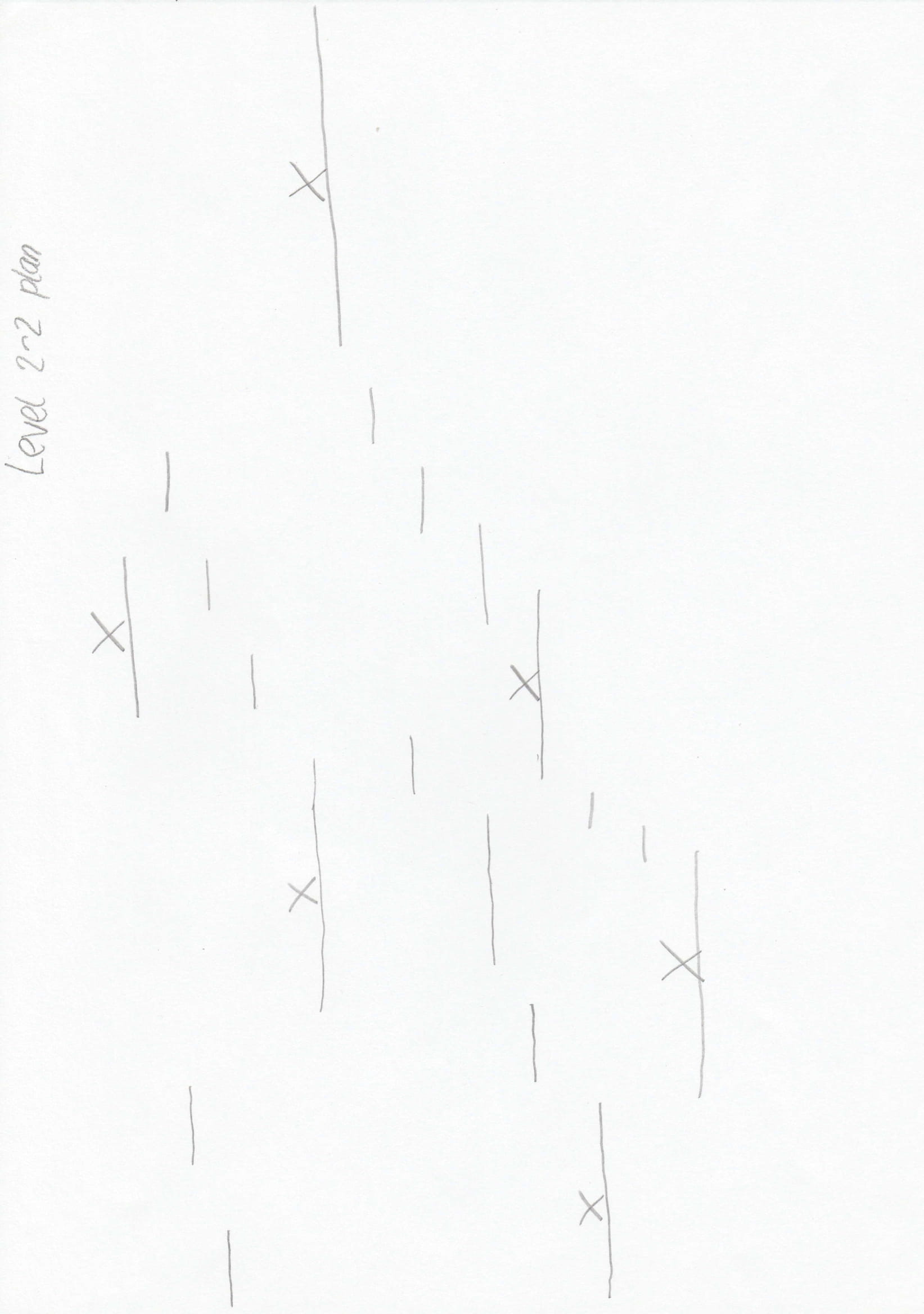
The project plan was able to be followed through the development despite the lack of use of the Gantt Chart. Some parts of the project did deviate from the plan. One of these deviations was the use of the Fungus engine. Despite only being planned to be used to create the dialogue, the Fungus tool was greatly used throughout the development of the game. Little changes like this were done throughout the project whenever an issue came up.

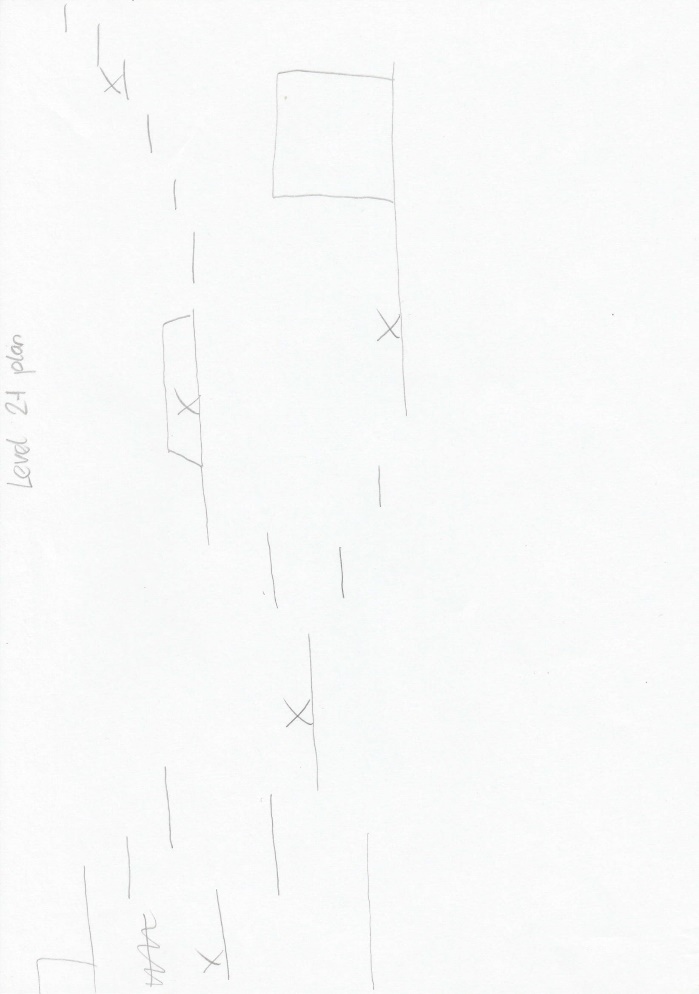
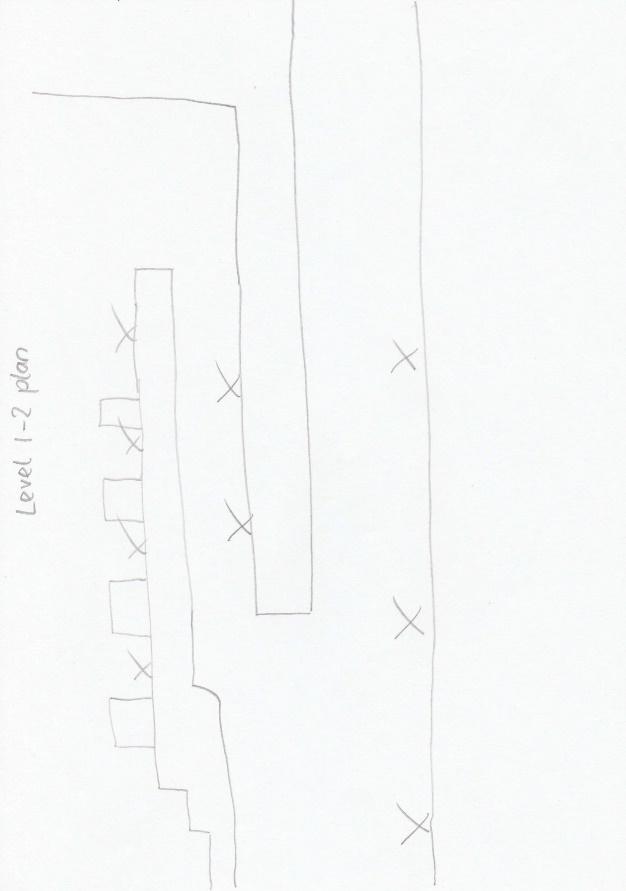
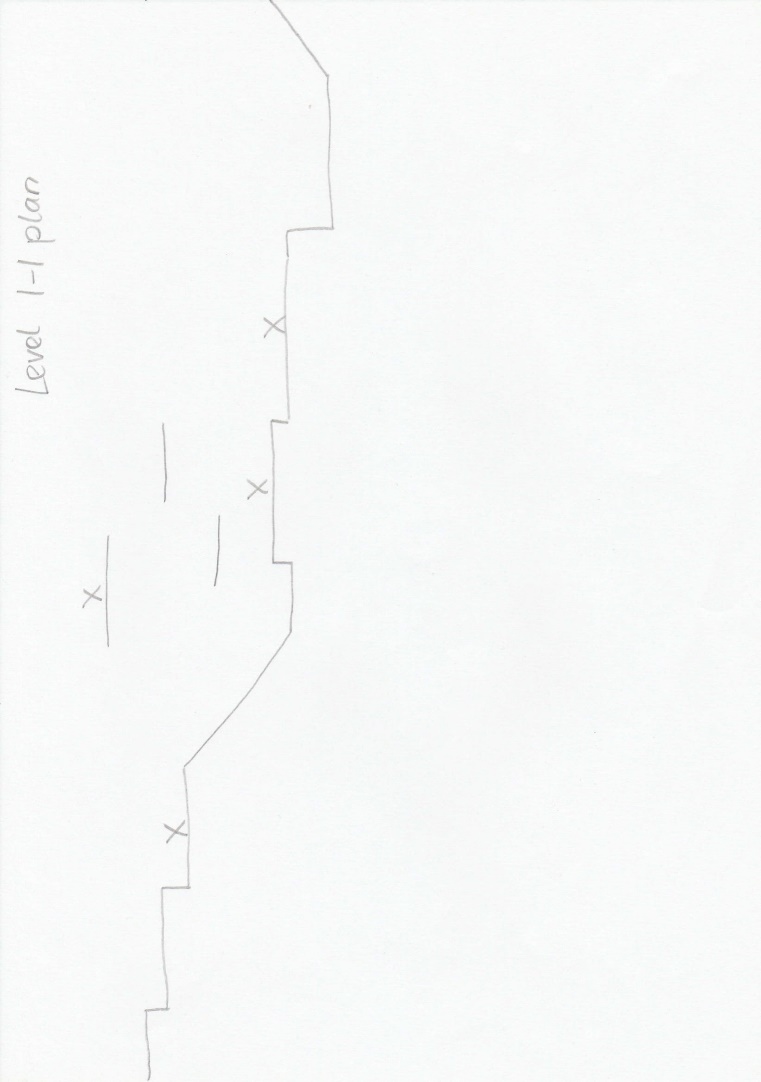
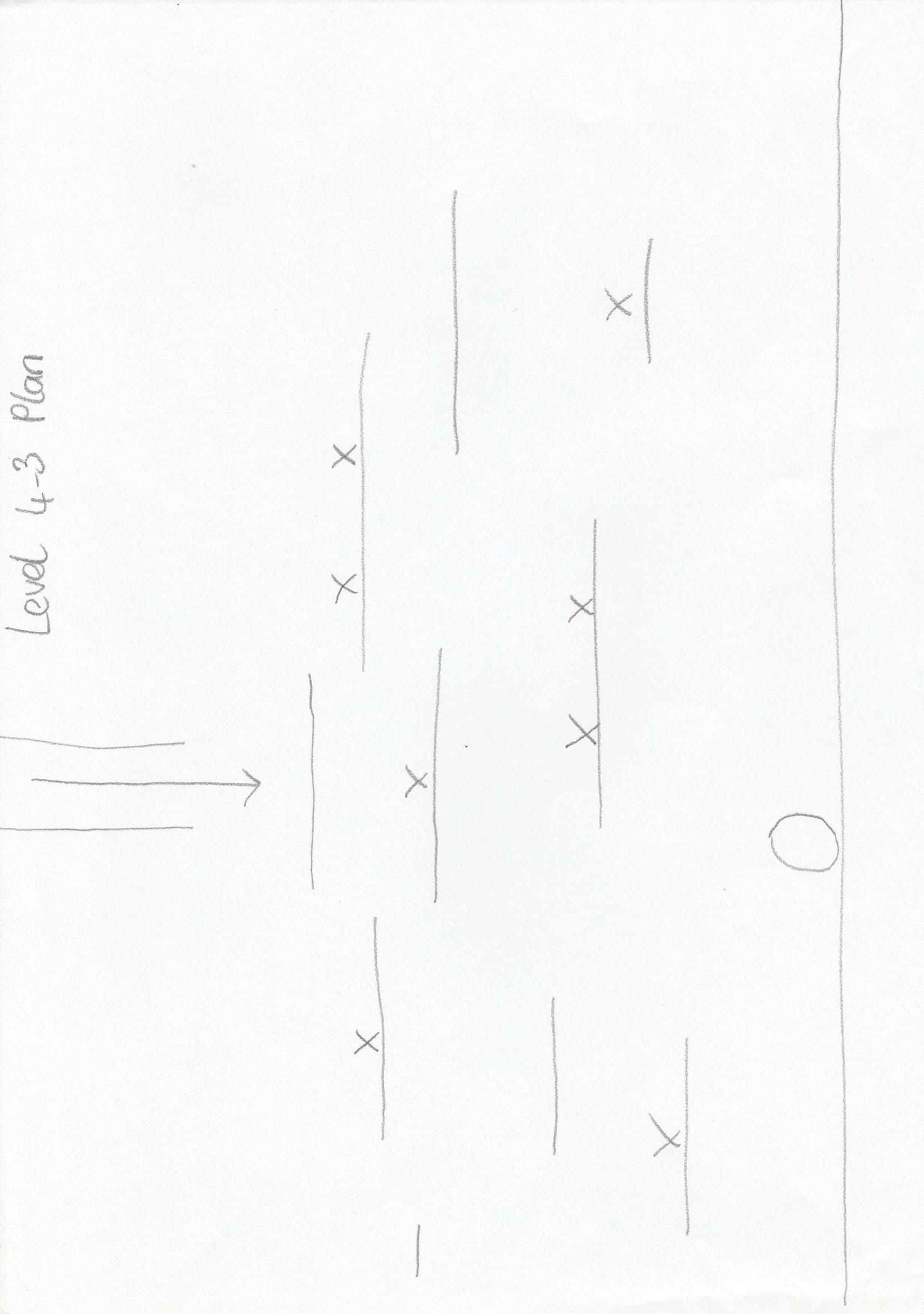
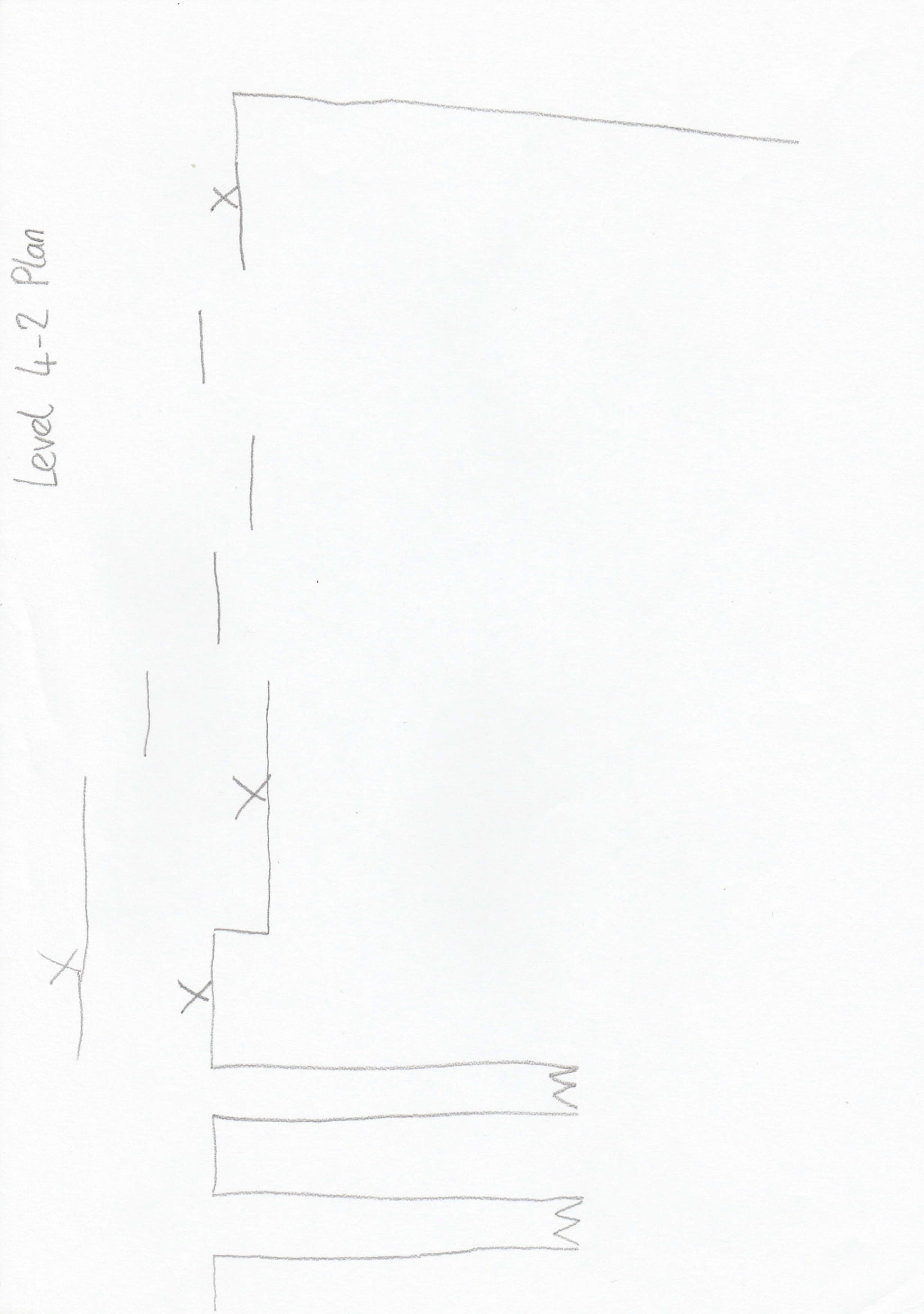
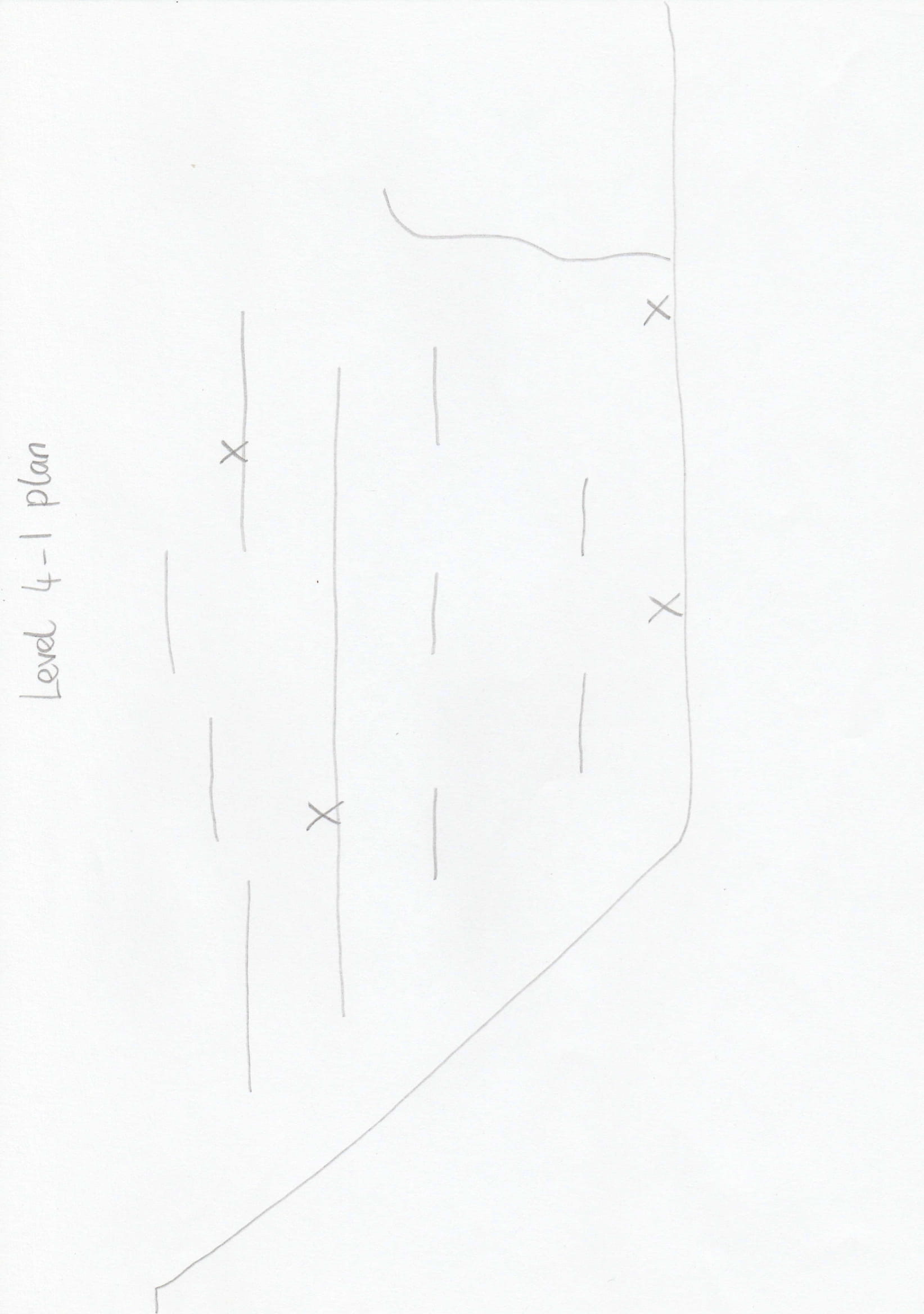
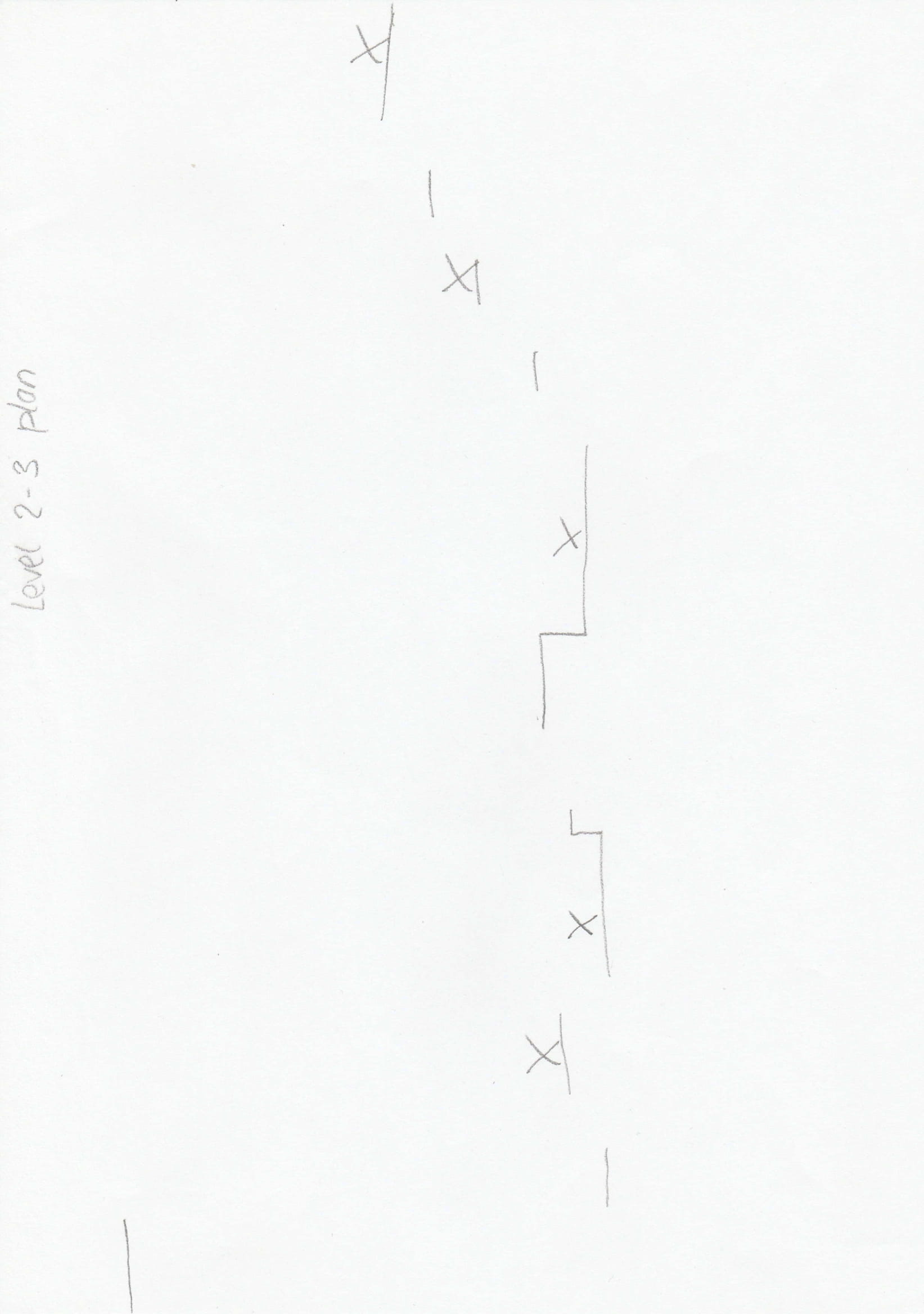
Despite some areas needing improvement and some functions having to be left out, the player’s choices were still able to be implemented, providing the player with a game that they can (and may need to) replay several times to see all the different outcomes and routes they can go down. This main function was vital as it was the main requirement of the project.

In conclusion, despite some slight deviations, a solid demo was able to be created. This demo shows the main functions that are required for the project such as various story paths, dialogue, platforming levels and boss battles.

# Appendix

## Level Plans





# References

*Boss Fight (1 of 2) - Unity 2D Tutorial (C#)*. 2015. [video] Wabble – Unity Tutorials.

*Boss Fight (2 of 2) - Unity 2D Tutorial (C#)*. 2015. [video] Wabble – Unity Tutorials.

*How to save and load player position 2D/3D - Unity Tutorial*. 2019. [DVD] ZCodes.

Makepixelart.com. n.d. *Make Pixel Art - Free!*. [online] Available at: <https://makepixelart.com/#> [Accessed 1 May 2020].