User Manual

Nam Le - jssb25 **Charles Dubois-Veltman - rklb65** **Joshua Pulham - tgbp44**

James Watson - nqcv52 **Sam Ezeh - vqqw43** **Akanksha Sirohia - nwrz52**

Link to final product: https://github.com/COMP2281/software-engineering-group-17/releases

Contents

1. Summary of document

This document is split up into six main sections beside this summary. Since it is a large document, we will be giving point of interests so that you can navigate this easier.

- **Section 2:** Summary of the project to contextualise the requirements and problems tackled.
- **Section 3.2:** Usable access to the system developed.
- **Section 3.1 and Section 3.3:** Description for each of the user requirements, including the initial and current status, the extent to which it is fulfilled, and justification for the extent of fulfilment.
- **Section 3.3.2:** Detail how the other non-functional aspects of the system are handled.
- **Section 4:** Source materials upon which the conceptualisation and development of the system is built
- **Section 5:** Jargon-free detail how to use the system from the perspective of non-technical users.
- **Section 6.1:** Technical detail how to use the system from the perspective of technical users.
- **Section 6.2:** Technical detail how each of the system's functionalities is developed.
- **Section 6.3:** Specifics of how the system can be maintained.
- **Section 6.4:** Possible future development that can be implemented.
- **Section 6.5:** Potential ethical and societal impacts of the system in its current and possible future status.
- **Section 7:** A glossary for terms specific to game development.

2. Project introduction

This is the user manual for IBM's "Reimagine Skills Build as an RPG game" project.

Skills Build is an educational website which provides free courses in multiple fields of technology such as AI, or cloud, and aims to develop premier skills for students on the platform. The courses also provide badges and certificates after completion, which can be displayed on LinkedIn or added to a student's CV to further their careers.

However, Skills Build in its current rendition is very complex to navigate and has some other issues. In particular, there is/are:

- No separation between a beginner course and an advanced course
- No way to retry quizzes and courses once completed
- Various ways of cheating the system and getting a badge without knowing the content
- Multiple websites used to host the Skills Build content

All these issues combined lead to students getting lost on the website(s), reducing the amount of knowledge acquired, and worsening the user experience in general.

The new system aims to gamify the learning experience, allowing for a more fun and engaging experience as students quest for in-game items relating to the badges, while still keeping the Q&A structure of the original quizzes. We include in-game links to Skills Build courses to deal with the complex-to-navigate issue, add preventative measures against cheating in the form of a boss fight, and allow for retrying courses and quizzes by presenting the player with the option of starting a new save file.

3. Solution discussion

3.1. Project Brief Analysis

Our solution has covered the main requirement from [LINK TO < Project-brief >]. So far, we have a functioning RPG game with items that can be acquired from beating the bosses through Q&A style combat in different worlds linked to the Skills Build badges.

Project description (please provide as much detail as possible, please include key performance indicators (KPIs)	**Skills Build is IBM's premier skills and career offering for students. It can however be complex to navigate, and students can lose their way. Can we create an RPG game world where Skills Build resources are linked in the game, and players are encouraged to quest for in game items, that are linked to successfully answering Q&A on topics taken from badges.** **Introduction to Threat Intelligence** **Introduction to Cloud** **Introduction to Data Science** **Introduction to AI** **Journey to Cloud** **And documentation from:** **z/os** **quantum** **Automation products** **Engineering products**
The challenge to be addressed	To create a 21st century learning game based on IBM Skills Build

Figure 1: Image of project description and challenge to be addressed in the project brief sent by the client.

Currently, two badges have been fully implemented in the game as worlds, those being "Introduction to AI" and "Introduction to Data Science". We planned to cover IBM Skills Build Courses "Intro to Cloud", "Journey to Cloud", and "Threat Intelligence and Hunting" in three other worlds, however, we were not able to finish these worlds due to time constraints and project down scoping.

Despite this, we have laid the foundation in the game so that adding the other three badges would be very simple. We also have concepts for the other three badges' world, which will be further discussed in Section 5.4.

Unfortunately, we did not add any of the documentation, as the resources could not be found on the IBM Skills Build website.

Regarding the "challenge to be addressed" section, we have done lots of research on existing 21st games (see Section 4) as well as existing learning games (see our requirement specification [1]), and have made design choices driven by the findings to make our game fit the "21st century learning game" demand.

3.2. Installation Guide

The following section will give a step-by-step guide on how to download and run the game in its latest build. The steps are as follows:

1. Head to the project's GitHub page (https://github.com/COMP2281/software-engineering-group-17/releases). Once on there, look for the release section as shown in Figure 2, and click on it. This will take you to our releases.

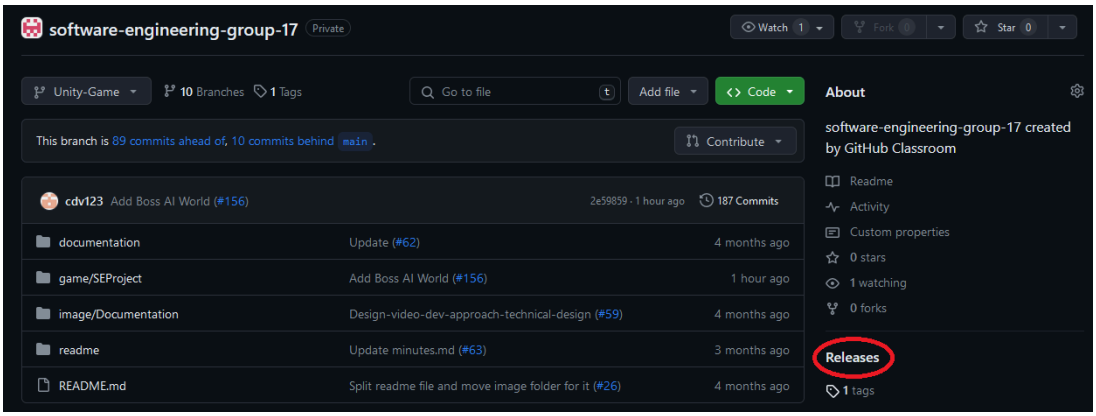

Figure 2: Image of the release section on the project GitHub page.

2. Once inside the releases, look for the newest version of the game. This should be the first card you see on the screen. Press on the "Assets" dropdown to show the files available, and press on the "rpg-skills-build" .zip to download the build.

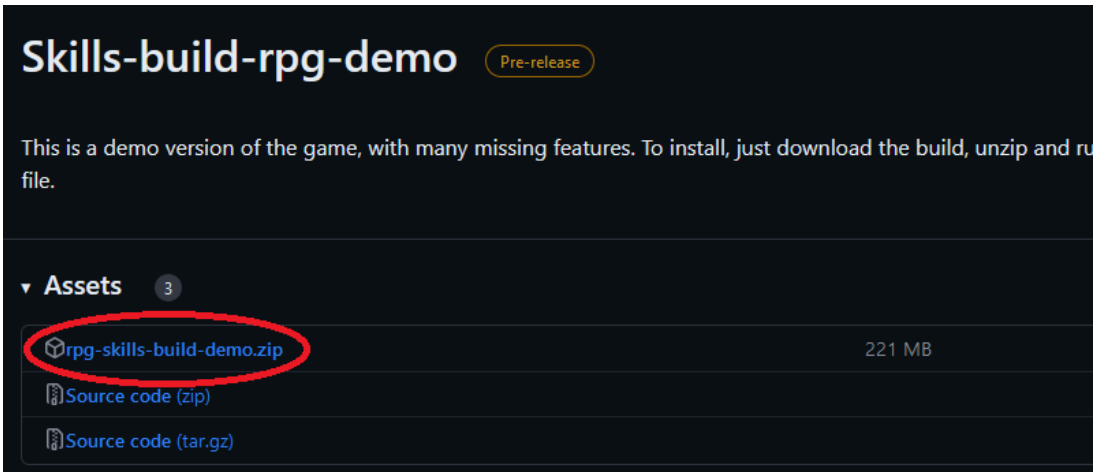

Figure 3: Image of the .zip build folder in the release section.

3. After downloading the game build folder, unzip it using your tool of choice. Once unzipping is done, open the folder and look for the "SEProject.exe" file, and run it. This should open the start screen of the game.

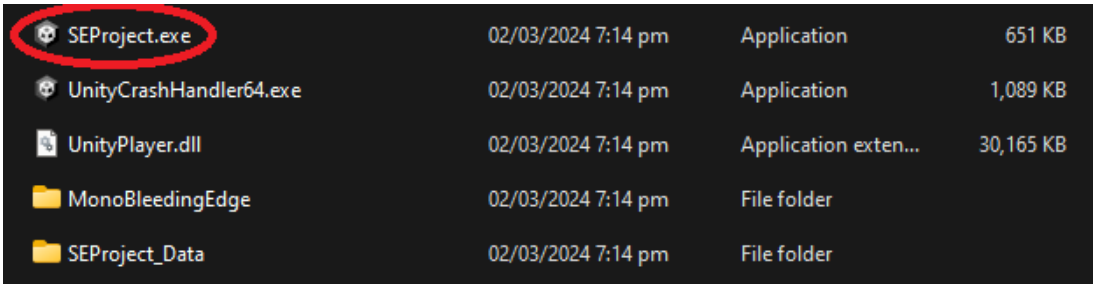

Figure 4: Image of the .exe file in the game build folder.

Note The game is only compatible with Windows 10 and 11 as per the request of the client. We recommend having a computer with a minimum of 4GBs of RAM to run the game. You can feel free to plug in a controller or play the game using the keyboard and mouse.

3.3. User Requirements Considerations

In this section, we refer back to the requirement specification and go through an analysis of functional and non-functional requirements in regard to our current solution. We consider their current status, the extent to which they have been fulfilled, and provide some additional justification for how we came to these conclusions.

3.3.1. Functional Requirements

Requirement Name	Current Status	Extent to which this is fulfilled	Justification of fulfilment
FR1.1 - Player Character	Unchanged	Implemented	The Player Character is an IBM employee and has distinct sprites in and out of combat.
FR1.2 - Player Character Movement	Dash added for AI World	Implemented + Additional Features Added	The player is able to move in all cardinal directions. One of the worlds (AI-World) has a dash feature which the player can use to get a very quick speed boost in the direction they are currently facing.
FR1.3 - Player-Based system-Controls	Added additional controls for actions in AI World	Implemented with additional controls	Player has a dedicated interact key as well as a secondary interact key for other abilities such as the Shield added in AI World.
FR2.1 - Game System - Main Menu	Unchanged	Implemented	The Main Menu contains all buttons specified: Make a new game, continue an ongoing session, open the options, quit and close the game window.
FR2.2 - Game System - Pause Menu	Unchanged	Implemented	The player is able to open the pause menu using either the Escape or Tab key and in the Pause Menu is able to access the inventory, save the game, open the options menu, or quit the game.
FR2.3 - Game System - Saving	Unchanged	Implemented	The player is able to save the game in the Pause Menu.
FR2.4 - Game System - Inventory	Unchanged	Implemented	The inventory exists in the Pause Menu and the player

			is able to view their collected items.
FR2.5 - Game System - Camera	Modified to have two types of cameras, cameras which stand still, and cameras which move with the player, this was done to provide a smoother experience, and is common for 2d top-down games	Implemented modified version	AI World and Data Science World use fixed camera and camera which follows the player, the Hub World only uses a camera which follows the player.
FR3.1 - Gameplay Systems - Combat	Modified - health bar replaced by a number of hearts	Implemented modified version as a number of hearts works better for answering questions, as answering wrongly deduces a fixed number of hearts	The game features a combat system where the player takes damage when attacked by an enemy, and dies when at 0 hearts.
FR3.2 - Gameplay Systems - Skills Build System	Modified - replaced skill tree with options which show up upon completing a world with different options available as to which power up to apply. This was done to increase simplicity for the user as well as for developing.	Implemented modified version	The Skill Tree permanently upgrades the player's abilities, but cannot be accessed in the pause menu.
FR4.1 - Gameplay - Central Hub	Unchanged	Implemented	The Hub World can be accessed and the player can enter all other worlds from it.
FR4.2 - Gameplay - Worlds	Unchanged	Implemented	Each world contains many rooms and ends with a boss fight.

FR4.3 - Gameplay - Rooms	Modified with a more accurate term 'Levels', where levels each have a puzzle that can span multiple rooms.	Implemented	Levels contains a puzzle or a boss with a door to the next level.
FR4.4 - Gameplay - Puzzles	Modified to include a greater variety of gameplay.	Implemented	All levels have puzzles. But puzzles have a wider variety of gameplay than "multiple choice questions disguised as a minigame"
FR4.5 - Gameplay - Multiple Choice Questions	Unchanged	Implemented	Questions from IBM Skills Build are included in the game.
FR4.6 - Boss Fight	Modified so that bosses include questions asked in the puzzles beforehand.	Implemented	Each finished world has a boss fight.
FR5.1 - Audio - SFX and BGM	Unchanged	Partially implemented	We have added SFX and BGM but currently they are grouped together (i.e. they cannot be modified individually).

3.3.2. Non-Functional Requirements

Requirement Name	Current Status	Extent to which this is fulfilled	Justification of fulfilment
NFR1.1 - Executable file	Unchanged	Satisfied	See Section 3.2
NFR1.2 - Platforms	Unchanged	Satisfied	See Section 3.2
NFR2.1 - Graphics	Unchanged	Satisfied	See Section 6.2.2.19
NFR3.1 - Response Time	Unchanged	Satisfied	After testing the response time using a recording software to measure the response time between button press and movement on the screen, the response time was less than 0.1 seconds.
NFR3.2 - Smooth Movement	Unchanged	Satisfied	After surveying multiple people after they have played our game, the consensus was that the player's movement in the game felt smooth.
NFR3.3 - Beginner Friendly	Unchanged	Satisfied	Although not specifically survey, all players that had little to no experience playing games did not have any issues understanding how to play our game, therefore we believe that this NFR is fulfilled.
NFR3.4 - Room Transitions	Unchanged	Not Satisfied	Due to time constraints, room transition effects were not added.
NFR3.5 - Boss Battles	Unchanged	Satisfied	After surveying multiple people after they have played our game, the consensus was that the boss battles were engaging.
NFR3.6 - Music	Unchanged	Satisfied	After surveying multiple people after they have played our game, the consensus was that

			the music fit each area of the game.
NFR3.7 - Progression	Unchanged	Satisfied	The game includes an aspect of progression where the player gains items that buff their stats.
NFR3.8 - Frame Rate	Newly Added NFR, states that the frame rate should be at least 30 frames per second at all times	Satisfied	Used AI-World (the most intensive section) to test out frame rate and found the frame rate to be 60 frames per second, this will need to be tested again once more sections are added and the game is further in the development process.
NFR3.9 - Volume	Newly Added NFR, audio levels should be at acceptable volume	Satisfied	All people who tested our game had no complaints about the audio levels, as the volume can be adjusted in the main/pause menu.

4. Conceptualisation and development

In this section, we will discuss the source materials upon which the conceptualisation and development of the system is built upon, and then show how we executed upon those ideas.

4.1. Market Research

RPGs are a genre of video game that has been continuously iterated upon since the 1970s [2], as such before starting development on our game, we decided to do market research into some pre-existing RPG titles, alongside some other games we felt were relevant to our game.

4.1.1. Combat

RPG combat has incredible variation, and there is far from one set presentation or style that must be followed for an RPG game, thus we need to decide on what format we want the question and answer (Q&A) combat to be in, firstly in how we want our player and our enemies to be represented in battle. Additionally, while there are many RPGs that are not turn-based, due to the Q&A nature of our combat, we looked predominantly at turn-based RPGs.

4.1.1.1. Undertale

One of the first RPGs that come to mind when we consider combat encounters that allow you to talk to your enemies is the 2015 game Undertale, which does not feature the player character and has a front shot of the enemy you are fighting or talking to.

Figure 5: Image showing a fight sequence in Undertale.

4.1.1.2. Final Fantasy VI

Alongside the likes of Dragon Quest, the Final Fantasy series is one of the forefathers of the RPG genre; Final Fantasy VI is the last 2D mainline Final Fantasy game, the game features combat where players and enemies face each other from opposite sides of the screen, with sprites that show their full bodies.

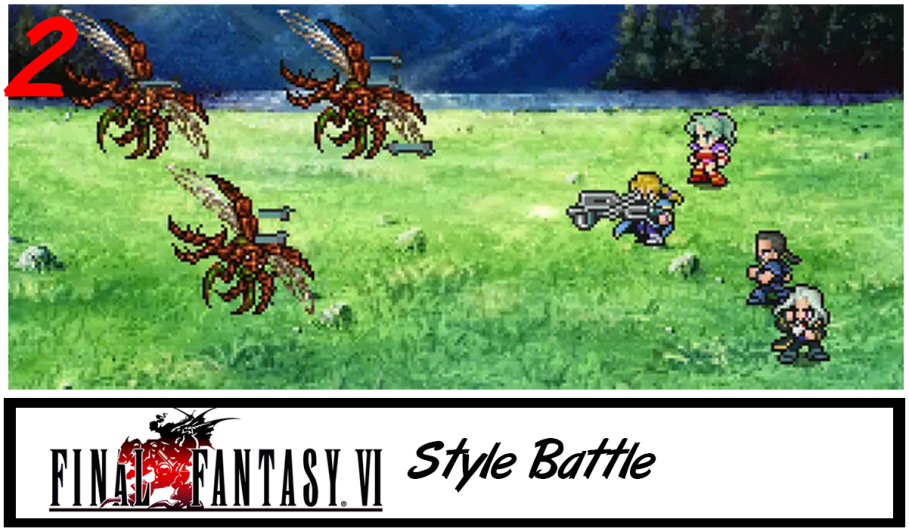

Figure 6: Image showing a fight sequence in Final Fantasy VI.

4.1.1.3. Pokémon

As the highest-grossing media franchise in the world[3], the original Pokémon games have captivated many people over the years, featuring two characters facing off on opposite corners, both with a 3/4 perspective sprite, with the player character facing away from the camera.

Figure 7: Image showing a fight sequence in Pokémon.

4.1.1.4. Ace Attorney Investigations

While not an RPG series, the Ace Attorney games are puzzle games, where the main "combat" encounters take place within a courtroom, featuring the attorney and prosecutor in a battle of wits. In the mainline games, these characters take up a full screen to themselves, however, the spinoff Ace Attorney Investigations features the two characters on one screen, battling with their words. Considering the dynamic expressions and how the game requires you to "talk to fight", I thought this would be a good fit for the style of game we are making.

Figure 8: Image showing a fight sequence in Ace Attorney.

4.1.2. Menu

We also need to consider how the player will answer questions, and what type of menu we want the player to use.

4.1.2.1. List Style

This is arguably the most popular style of menu in RPG games and involves a scrollable list, to which players will move down the list, relatively simple to implement and does what it needs to.

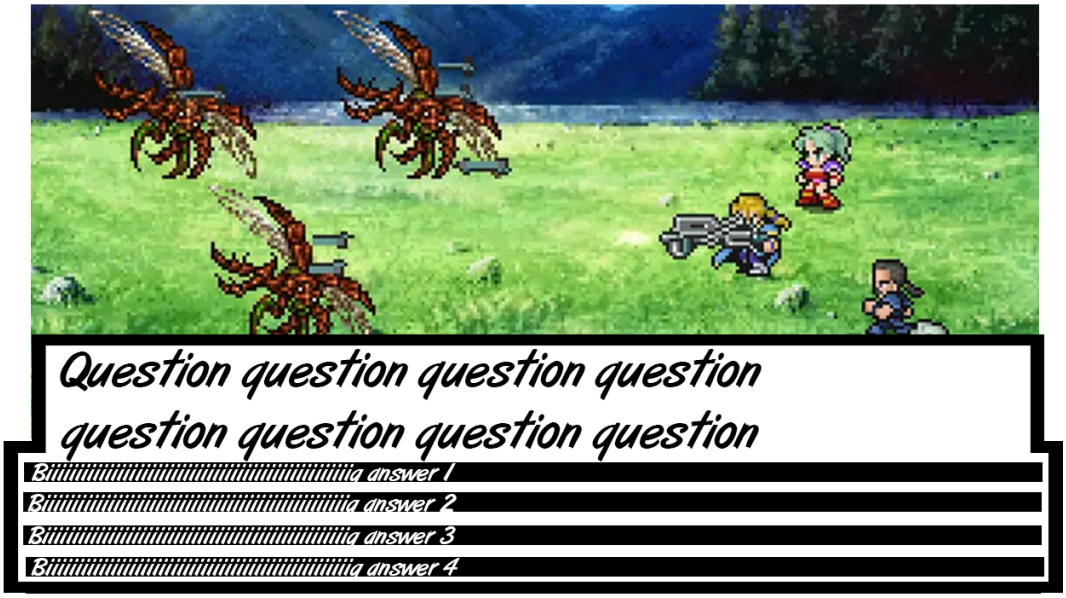

Figure 9: Image showing the list menu style in Final Fantasy.

4.1.2.2. Square Style

Having a square menu style with 4 options is less popular in the genre, but is still featured in the monolithic franchise Pokemon, this style may work quite well for Q&A style combat, however is slightly more difficult to implement, and limits us to 4 answers.

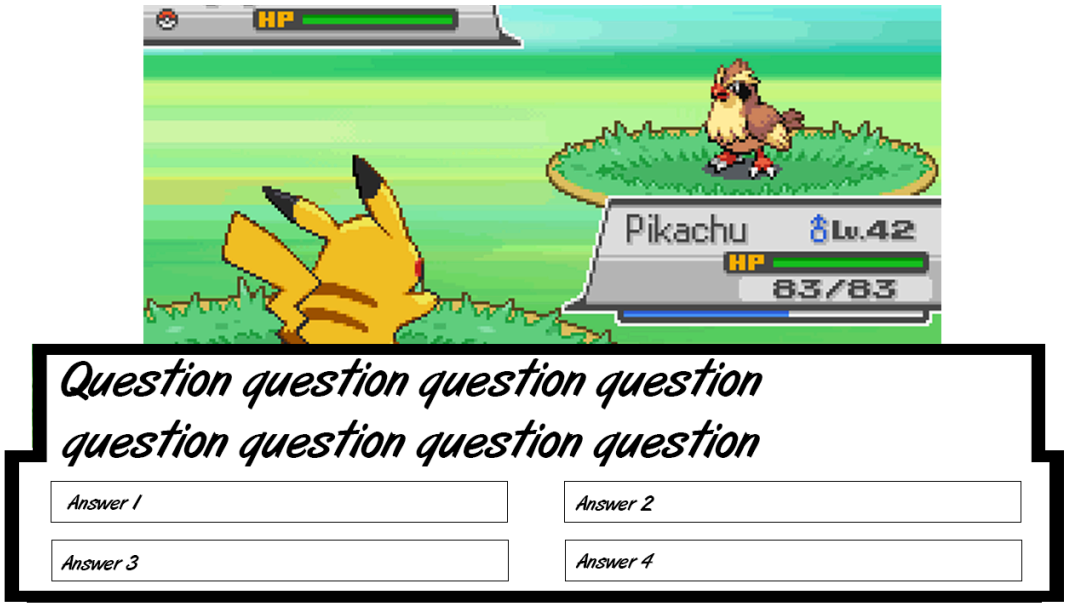

Figure 10: Image showing the list menu style in Pokémon.

4.1.3. Sprites

While combat sprites depend on the combat style we choose, all the characters in the game need to have consistent sprites in the overworld, and so we looked into the different sprite sheets used in various other RPG games, both from a complexity standpoint, due to our limited knowledge of art, and stylistic standpoint

One of the biggest things to consider with sprite sheets is the amount of space they take up relative to the world around them, traditional RPG style has the world split into “tiles”, which can be thought of as a square grid, in which the characters normally move between these tiles, and normally mean that sprites have to fit into that square area. However, newer top-down 2D games are less restrictive, and have more free movement, which also means that character sprites do not have to conform to set tile space

The sprites I am showing off in this section are all walking animation sprites, with a step-stand-step style of animation, which is simple to animate, but works brilliantly, and is still used in modern top-down games today.

Figure 11: Image showing nine different sprite styles belonging to different games

4.1.3.1. Final Fantasy VI

Final Fantasy 6 sprites differ from the rest as they only have 2 frames on their sideways walk, they are slightly taller than a single tile but would still work for a tile-based game, and are relatively simple. And considering the person doing the art for our group does not have a great deal of experience, simple sprites are probably better.

4.1.3.2. Fear and Hunger

Fear and Hunger sprites are 2 tiles tall, which means that while their hitbox is in their lower half, they also have an upper half, which lets them appear much more human and realistic. These sprites are much more complicated but look amazing because of it.

4.1.3.3. Stardew Valley

Stardew Valley's sprites do not conform to the tile system, but character designs are still simple, yet appear to be humanlike by not conforming to tiles.

4.1.3.4. Pokemon

Pokemon games are the classic representation of tile-based sprite sheets, and while simple and stylised, they also carry a lot of detail.

4.1.3.5. RPGMaker

RPGMaker is a game development engine inspired by classic RPG games, and thus operates off a tile system, with their example sprites all fitting into one tile.

4.1.3.6. Simple

Finally, we have the super simple 16x16 sprites, which conform to tiles, but lack a lot of detail.

4.1.4. Health

We need to be able to keep track of how many questions have been answered correctly and there are several ways to do this, however keeping in line with RPG games, the best way is through Hit Points (HP), which normally represent how much damage the player has taken [CHECK DEFINITION]. Converting this to Q&A style combat, we can say that if the player answers a question correctly, the enemy takes damage, and if you answer a question wrong, the player character takes damage.

There are multiple ways of representing HP, however we will be deciding between health bars and heart bars.

4.1.4.1. Health Bars

A health bar is quite simply a bar to display health which is normally analogue. We take an example from Guilty Gear Xrd, a fighting game released in 2014. As this health bar depletes it will "empty" towards the centre of the screen.

Figure 12: Image of combat in Guilty Gear Xrd where both characters have full health.

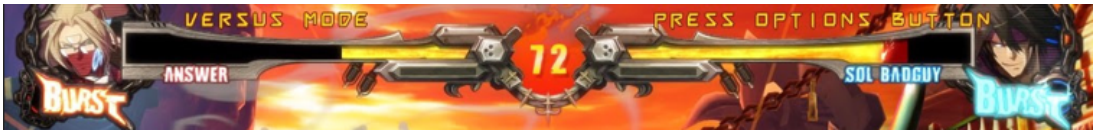

Figure 13: Image of combat in Guilty Gear Xrd where both characters are missing health.

Or take another example from Trails from Zero, an RPG originally released in 2010, which features 4 "party members" each of which having 3 bars, "Health Points", "Energy Points" and "Craft Points", each of which depletes towards the left as you use them or are damaged. In addition, if you bring your Craft Points over 100, there is another bar, in a lighter colour of green, that overlaps the original bar, which is commonly used in games to represent a strong character having multiple health bars.

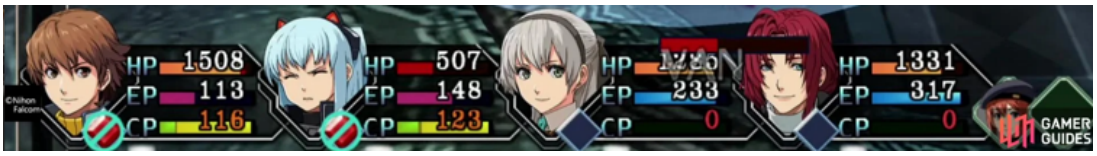

Figure 14: Image of combat in Trails from Zero, showing multiple stat bars.

The advantage of health bars is that they are simple and easy to look at, as they can be represented numerically, and thus can be manipulated freely.

4.1.4.2. Heart Bars

Heart bars are a more discrete representation of health bars, as instead of having a large number to represent health, you instead have a more constant smaller numerical representation, and each piece of damage would represent one “heart”.

In Minecraft, you have 10 hearts, where each piece of damage counts for a quarter of a heart, thus giving a visual representation of 40 hit points.

Figure 15: Image of heart HUD in Minecraft, with the player having 7.5 hearts.

The positives of heart bars are they also allow you to represent health in unique ways, for example in Hollow Knight, each piece of health is represented by a “mask”, thus this concept allows for more creativity, as we can represent health as whichever item we desire, and we can still have as many hearts as we need bosses to have, by giving them multiple sets of hearts, however, this would require a lot more programming and asset creation.

Figure 16: Image of heart HUD in Hollow Knight, with the player having 5 hearts.

4.2. Current Implementation

From the above market research, we decided to model the combat on Ace Attorney Investigations, with 2 big sprites of the characters on each side, the menu used will be list style and the character sprites will be modeled on Pokemon characters, and the health system will be based on hearts.

4.2.1. Game Design Philosophy

After looking into the different possibilities discussed above, we had to decide how we wanted to implement them in our game design. The biggest inspirations for how we wanted to design the game overworld sections were two indie games named Undertale and Deltarune, primarily due to them being top-down 2D Pixel Art RPG games like ours will be, alongside the bulk of the game being interesting puzzles, broken up by fun combat mechanics. Overall it is a simple but very effective game design, making it perfect for this project.

Figure 17: Image of Deltarune.

Furthermore, we needed to figure out how we wanted to split the game up, as we wanted each section of the game to correspond with a section of IBM Skills Build. Therefore, we decided to connect levels through a “Hub World”, which is a world separate from the main levels, there to allow you to access individual levels.

Figure 18: Image of the Hub World in Crash Bandicoot: The Wrath of Cortex.

This was predominantly inspired by Crash Bandicoot: The Wrath of Cortex, due to its visually pleasing Hub World, and interactable NPCs. This concept fits nicely with the user requirements, as well as allowing the developers to work on separate worlds in parallel, without disrupting other progress.

4.2.2. Overworld Sprites

Following the style guidelines of Pokémon for the overworld sprites, our artist made a set of walking sprites for the main character, as well as standing sprites for each of the bosses, for the player to interact with for boss battles, and in the Hub World.

Figure 19: Image of main character's full walk cycle.

Figure 20: Image of all the boss' overworld sprites.

4.2.3. Combat Sprites

As we decided to base our combat around the Pheonix Wright Ace Attorney games, it felt only logical to use similar sprites for our combat as well, as such we decided to make our main character have a similar base to Ace Attorney's main protagonist, Pheonix.

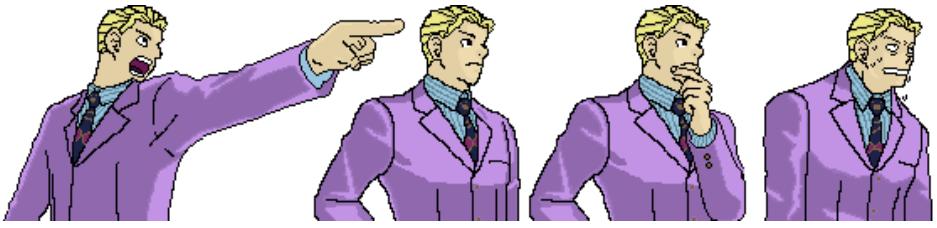

Figure 21: Image of the sprites used for the main character in combat.

For the boss sprites, we used a similar base to Ace Attorney's main antagonist, Miles Edgeworth.

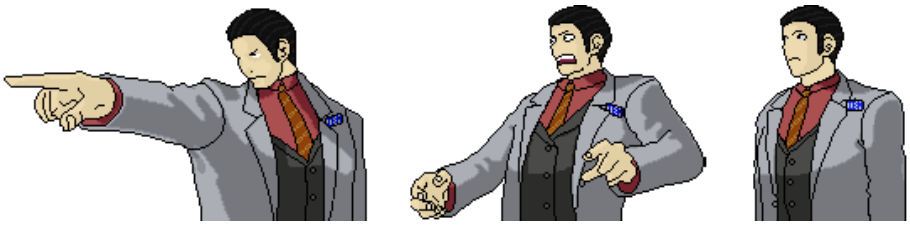

Figure 22: Image of the sprites used for one of the bosses in combat.

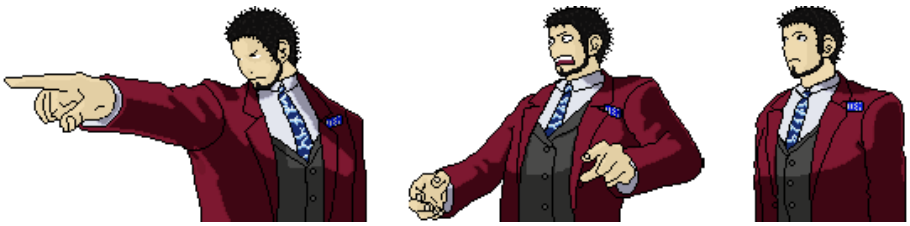

Figure 23: Image of the sprites used for another of the bosses in combat.

For our final boss, we decided early on in development that we wanted to base our final boss on a cyborg version of John McNamara.

Figure 24: Image of the sprites used for cyborg John McNamara in combat.

Lastly, we have two bosses who we didn't implement sprites for, the second of which is based on Mia Fey, also from the Ace Attorney games.

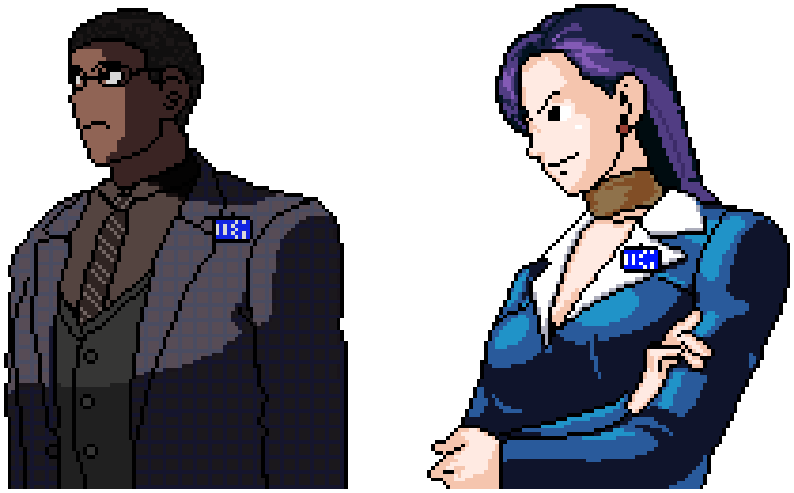

Figure 25: Image of concept work for the two unimplemented bosses.

4.2.4. Combat Menu

Other than sprite choices, the other major combat design choices we had to make were on the menu and the health, as well as the general presentation.

For the start of the battle, we decided to have the two characters point at each other, initiating the fight, before switching to the main combat section. An additional point to make is the combat backgrounds, we decided to use generative AI to produce backgrounds for combat, themed around our worlds, due to the speed and quality that they can be produced.

Figure 26: Image of the opening to a combat section.

After combat has fully started, the characters return to their neutral position, the question appears in the blue box, and the possible answers are in Final Fantasy list style in the box below it. You can tell which item you have highlighted by the red box, and this can be changed by moving up or down.

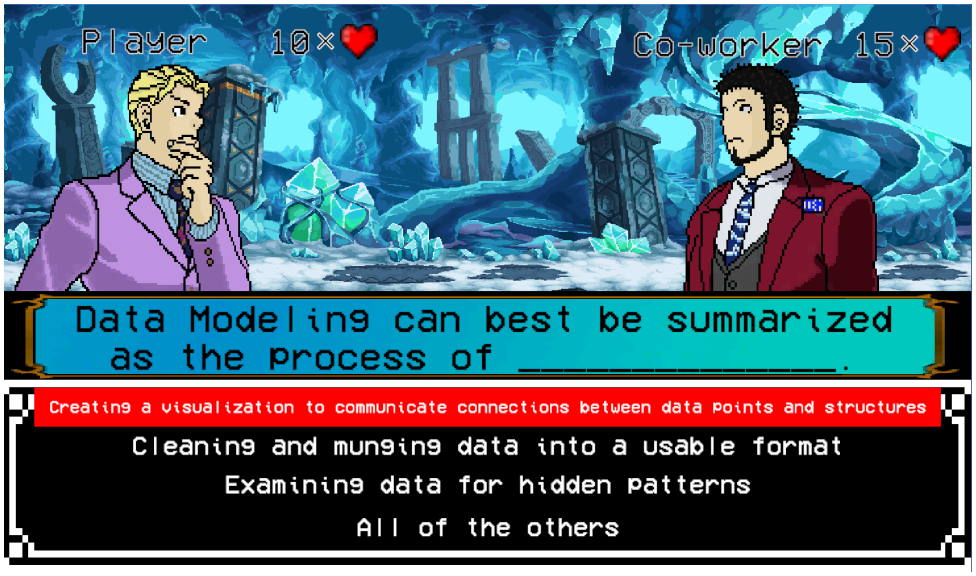

Figure 27: Image of the regular combat menu.

Upon answering the question correctly, the sprites change to their relevant states, and the boss loses health.

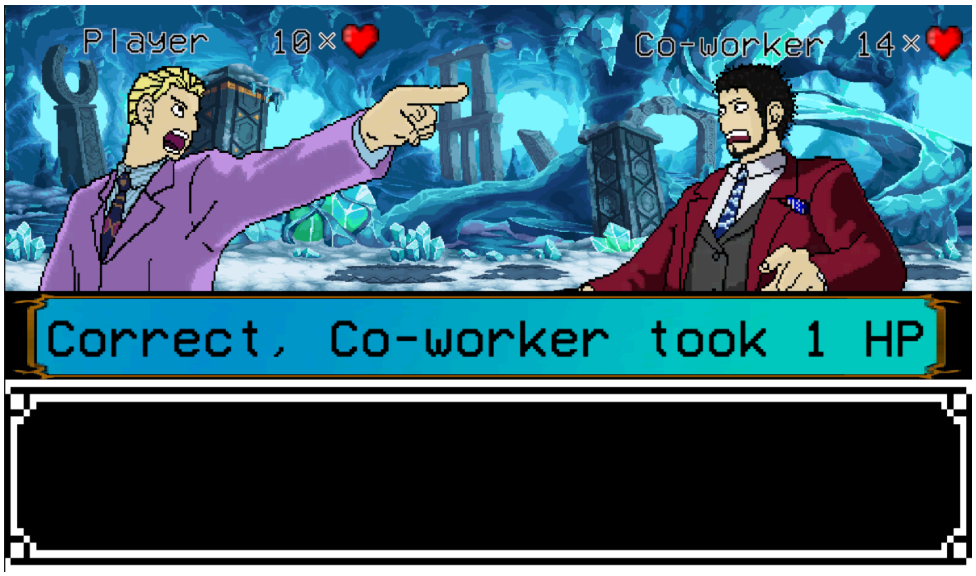

Figure 28: Image of the player answering a question correctly.

Upon answering incorrectly, the sprites once again change, but this time the player loses health.

Figure 29: Image of the player answering incorrectly.

Additionally, due to time constraints, we weren't able to implement true heart bars, as such they simply display the heart and the number of hearts for now.

5. Non-technical manual

The structure of the non-technical manual is inspired by a manual from a Nintendo title called “Pokémon Mystery Dungeon: Gates to Infinity” [4].

In this section, we refer to the user in the 2nd person format, addressing them with “you” and “your”. We also use less formal language and terminology. This is because the document aims to talk the user through the game as if they themselves were playing and experiencing it.

This document will be included alongside the final game build as a kind of game introduction booklet. The presentation will of course be more bright and lively compared to how it is shown in this document, including custom-crafted backgrounds, less formal fonts, decorated tables and image positioning, etc.

5.1. Getting Started

5.1.1. Introduction

Our game’s story begins with our main character, an IBM employee referred to simply as “the Protagonist”, doing work in their office like any other day when they are suddenly sucked into their computer and transported to the world of the IBM Tower of Skills Build Mastery.

Figure 30: Image of the Protagonist sprite thinking.

Dazed and confused, the Protagonist wakes up in this world and decides to explore their new surroundings when suddenly, they are beckoned to an altar by an unknown voice.

When arriving at the altar, they are greeted by the spirit of the altar, a helpful character that gives the Protagonist insight into where they are and what they must do to leave.

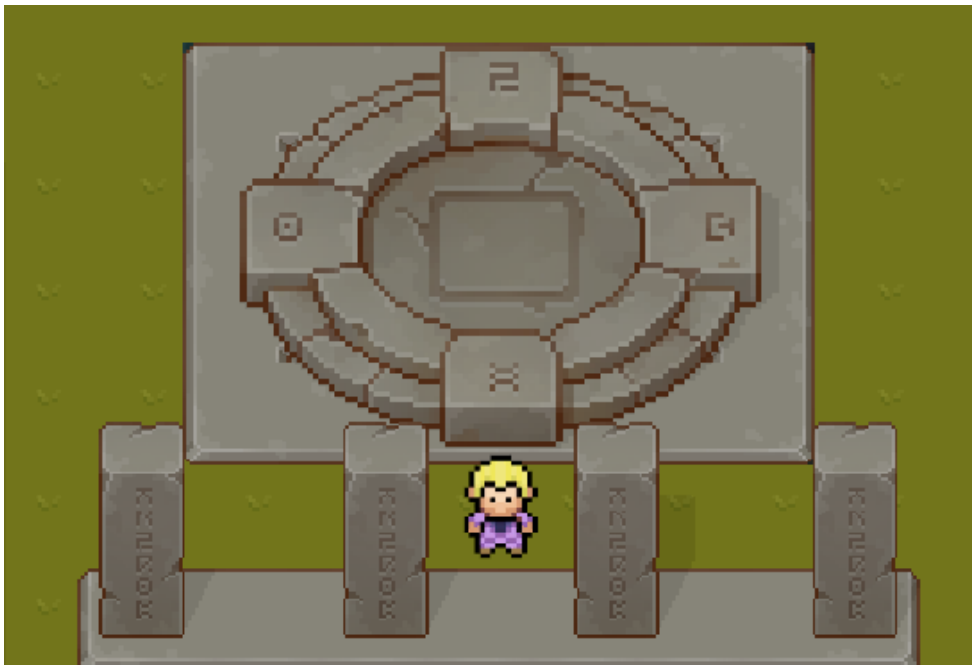

Figure 31: Image of the Protagonist in front of the Altar.

The world within the computer exists between time and space and needs to disappear for the real world to continue functioning. Destroying the world is also the only way for the Protagonist to go back home, so they agree to help the spirit in destroying it.

The spirit instructs the Protagonist to find 4 relics of alternate worlds and bring them back to the altar. Only then can they reach the creator of the world and defeat him to destroy the world.

Armed with this knowledge, the Protagonist embarks on a journey through each of the worlds, solving puzzles, playing mini-games, answering questions from Skills Build and fighting battles of wits against corrupted IBM co-workers who have also been trapped in the Tower of Skills Build Mastery.

Figure 32: Image of the Protagonist in front of a corrupted coworker.

Will you succeed in saving everyone and help them return to their original world?

5.1.2. Main Menu

The Main Menu is an inviting location, with sombre music and a futuristic background image with the IBM logo in it. The lively Menus background image moves around at random intervals in time, and blurs and unblurs itself too.

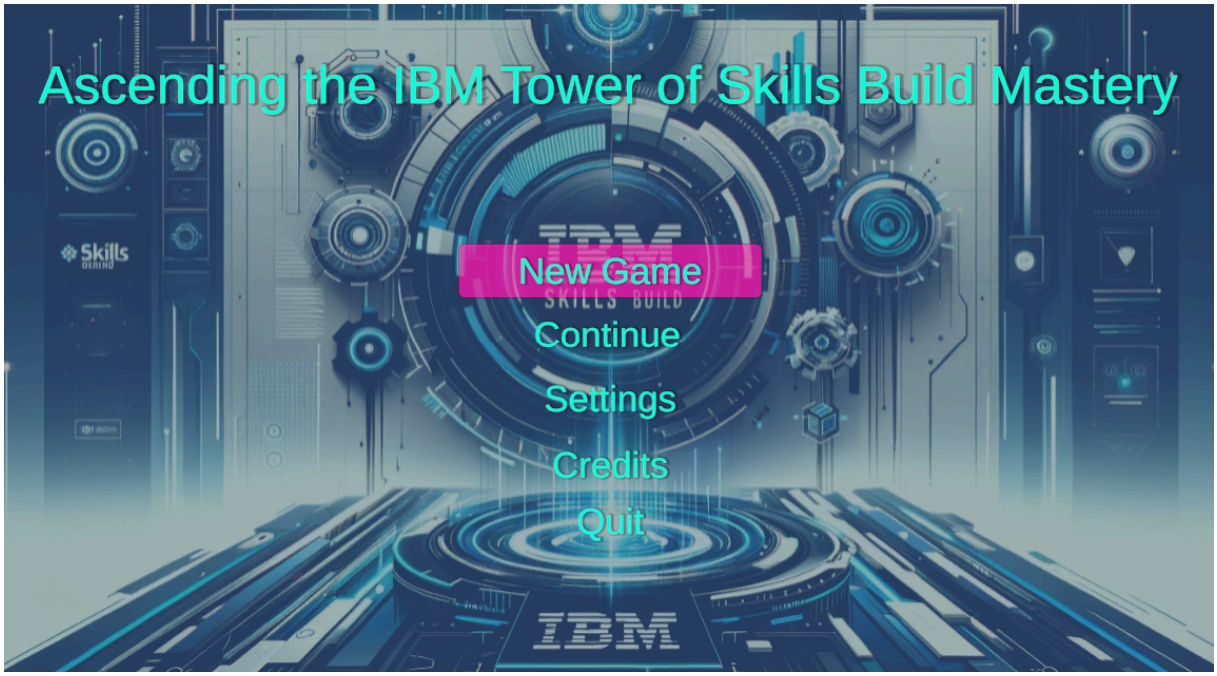

Figure 33: Image of Main Menu including background image and all buttons, with the “New Game” button being highlighted.

Main Menu Button	Description
New game	The new game button starts creates a new save file and loads the first level of the game.
Continue	The continue button loads the last save file and loads the level that was last saved.
Settings	The settings button opens the settings menu.
Credits	The credits button loads the credits for the game.
Quit	The quit button closes the game.

Table 1: Table of buttons for the Main Menu

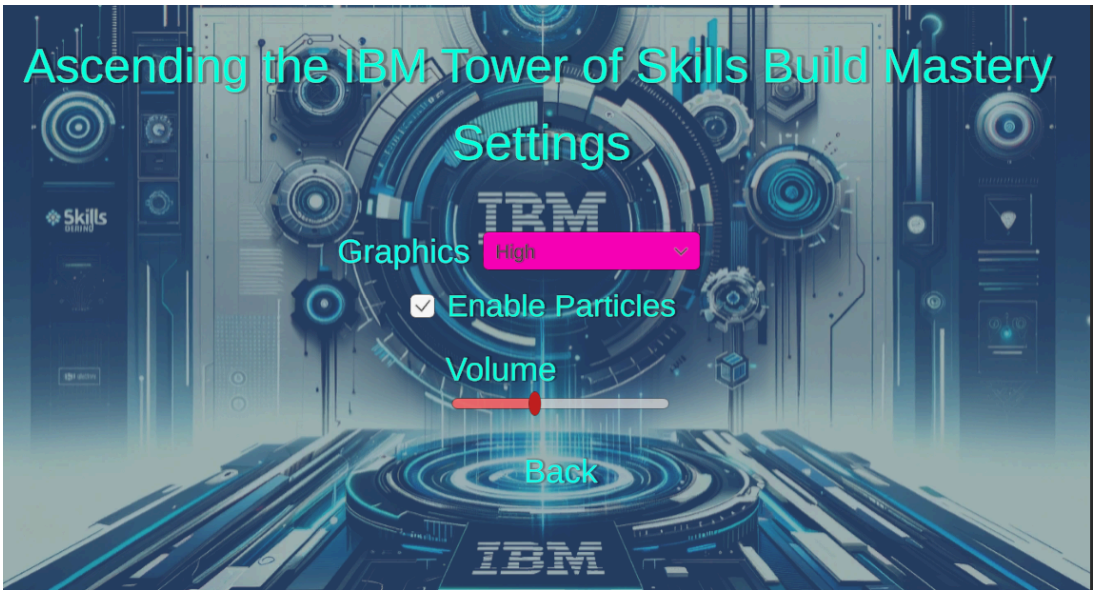

Figure 34: Image of Settings menu including background image and all buttons.

Settings Menu Button	Description
Graphics	The graphics dropdown allows you to change the graphical quality of the game. The 3 options are high, medium or low.
Enable Particles	The enable particles checkbox allows you to turn on and off particle effects in the game.
Volume	The volume slider allows you to alter the volume of the game.
Back	The back button takes you to the Main Menu.

Table 2: Table of buttons for the Pause Menu

Now that you've seen all that we have to offer in the Main Menu, let us explore The *IBM Tower of Skills Build Mastery* by starting a New Game!

5.2. Controls

This game is designed to be played using either a gamepad or a keyboard and mouse. While the game should be compatible with most gamepads, it has been designed primarily for the Playstation DualSense™ Controllers, the Xbox Wireless Controllers and the Nintendo Switch Pro Controllers.

As each of these controllers has a different layout, below is a graphic that can be mapped onto any controller, used for our control descriptions, for keyboard controls we will be referencing the key that needs to be pressed.

For the controls below, Left/Right Stick Up/Down/Left/Right refers to pushing the Left/Right stick in the specified direction enough for the game to register it as an input. Similarly, for D-Pad Up/Down/Left/Right and the other buttons listed, these require pushing the button enough for the game to register this as an input.

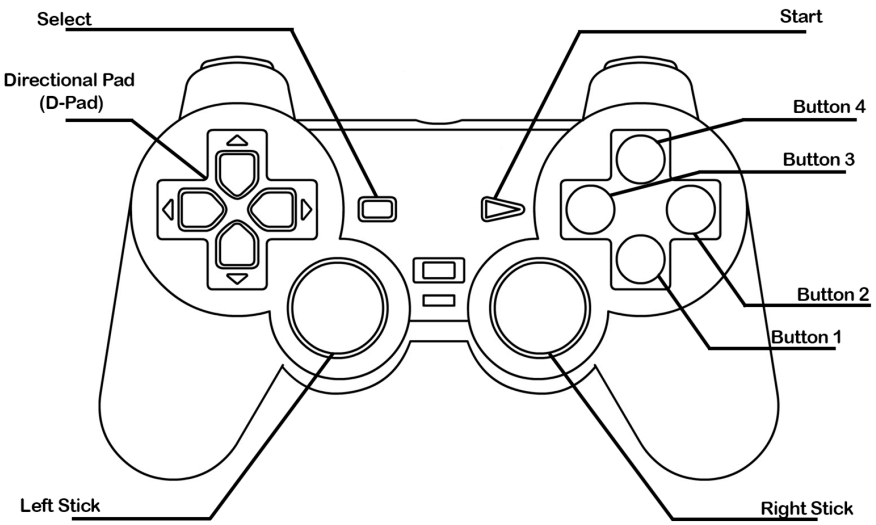

Figure 35: Image of a generic Gamepad with Button Labels [5].

5.2.1. Menu

Keyboard	Controller	Action
W / Arrow Up	Left Stick Up / D-Pad Up	Navigate menu up.
S / Arrow Down	Left Stick Down / D-Pad Down	Navigate menu down.
A / Arrow Left	Left Stick Left / D-Pad Left	Adjust menu item left.
D / Arrow Right	Left Stick Right / D-Pad Right	Adjust menu item right.
Space	Button 1	Select highlighted menu item.

Table 3: Table of controls for menus.

5.2.2. Overworld

Keyboard	Controller	Action
W / Arrow Up	Left Stick Up / D-Pad Up	The Protagonist walks upwards.
A / Arrow Left	Left Stick Left / D-Pad Left	The Protagonist walks left.
S / Arrow Down	Left Stick Down / D-Pad Down	The Protagonist walks downwards.
D / Arrow Right	Left Stick Right / D-Pad Right	The Protagonist walks right.
Space / Z	Button 1	Interact / Advance text.
Escape / Tab	Start	Enter Pause Menu.

Table 4: Table of controls for the overworld.

5.2.3. Battle

Keyboard	Controller	Action
W / Arrow Up	Left Stick Up / D-Pad Up	Navigate menu up.
S / Arrow Down	Left Stick Down / D-Pad Down	Navigate menu down.
Space	Button 1	Accept answer/choice / Advance text.

Table 5: Table of controls for battles

5.2.4. World Specific Mappings

5.2.4.1. AI World

Keyboard	Controller	Action
Space	Button 1	Dash.
Left Shift	Button 2	Shield.

Table 6: Table of controls specifically for the AI World.

5.3. Taking a break

Need a break from the game? Have something unexpectedly occur whilst playing that needs your attention? Worry not because we have added a Pause Menu!

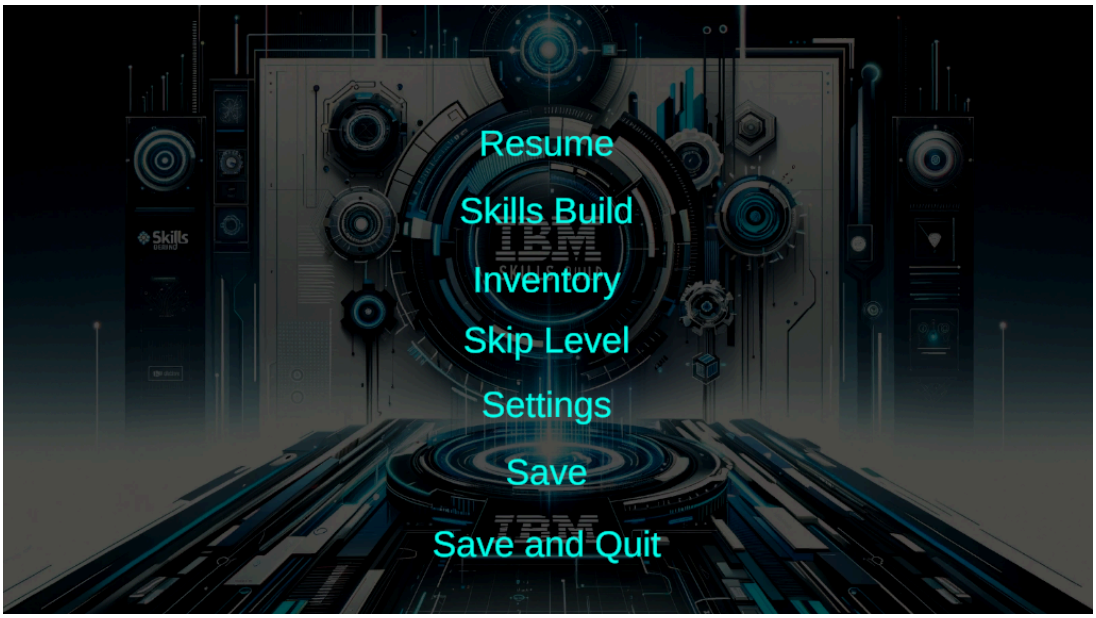

Figure 36: Image of Pause Menu including background image and all buttons.

Upon entering the Pause Menu, you will see a familiar sight; the same background image as the Main Menu! This is intentional, as both of the menus are designed to be welcoming and relaxing. We want the Pause Menu to be a safe retreat whenever needed, which is why the game is paused for the duration, so there is no need to worry about dying whilst in here.

Pause Menu Button	Description
Resume	The resume button resumes the game.
Skills Build	The Skills Build button opens a link to the Skills Build website, in case you would like to review the content there.
Inventory	The inventory button opens your inventory and shows you which relic you have acquired so far in your journey.
Skip Level	The skip level button will bring you to the start of the next floor. It is especially useful if you are struggling in the current level's puzzle.
Settings	The settings button opens the settings menu.
Save	The save button saves your progress up to the current level.
Save and Quit	The save and quit button saves the game, then opens the Main Menu.

Table 7: Table of all the functions of the pause menuy

The Settings Menu in the Pause Menu is the same as in the Main Menu, containing the same buttons and layout.

Skipping a level cannot be done in a Boss world or in the Hub World, meaning that you must beat the boss of each world to progress.

When saving the game, any ongoing progress within the current level is not saved, meaning that if you decided to close the game after saving, you would have to redo everything in the current level.

Make sure to reach the next floor before quitting the game!

5.4. Worlds

The game takes place in multiple worlds, each being a collection of levels that represent a different section of IBM Skills Build with its own themes and challenges. The only exception to this is the Hub World and the Final World.

5.4.1. Hub World

The Hub World exists to connect each of the worlds together, as well as being a safe location for the Protagonist to reside and call home. It is the central location to access all other worlds.

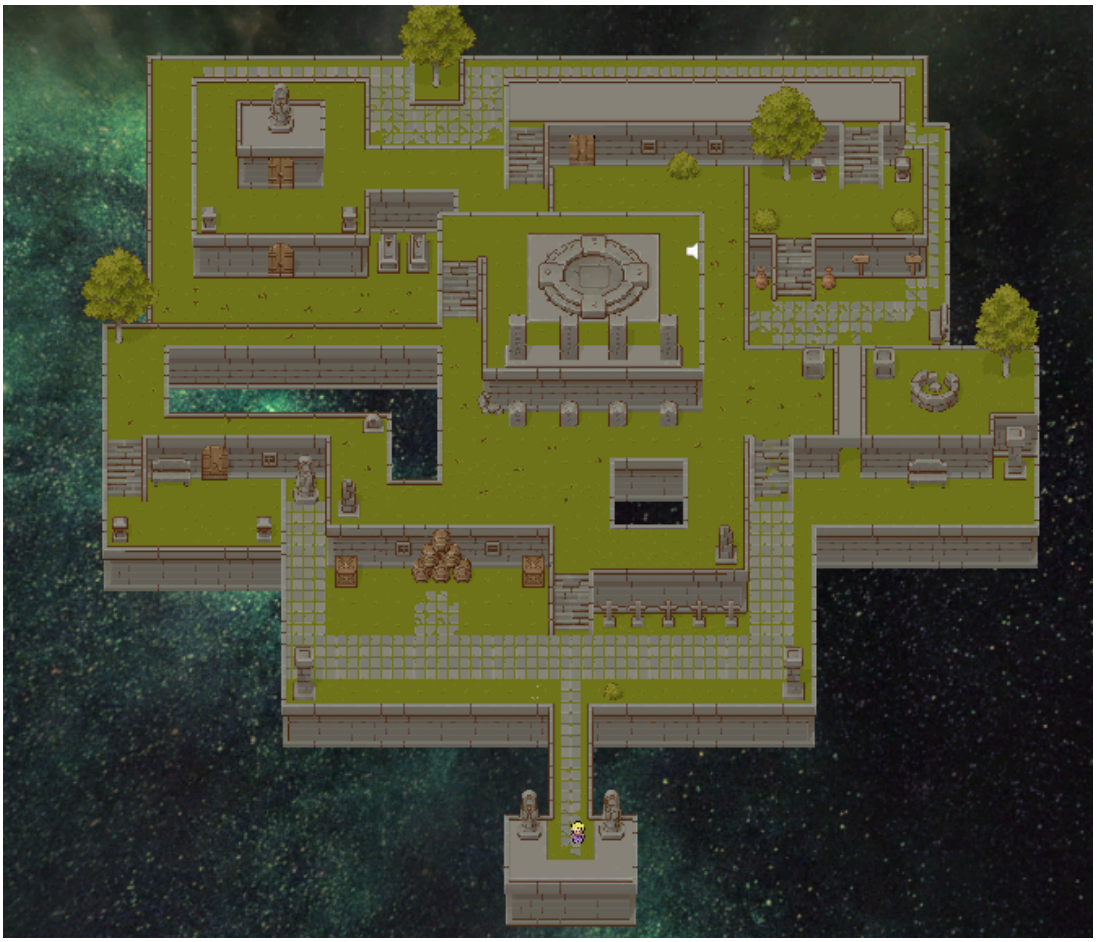

Figure 37: Image showing a bird's-eye view of the entire Hub World, with the Protagonist located at the bottom and the altar located in the center.

Here, the Protagonist will speak to the Spirit of the Altar for the first time, and be given their mission. All the rooms containing magic portals that connect to the other major worlds will also be opened after this first encounter, enabling you to choose whichever world you would like to enter, in any order!

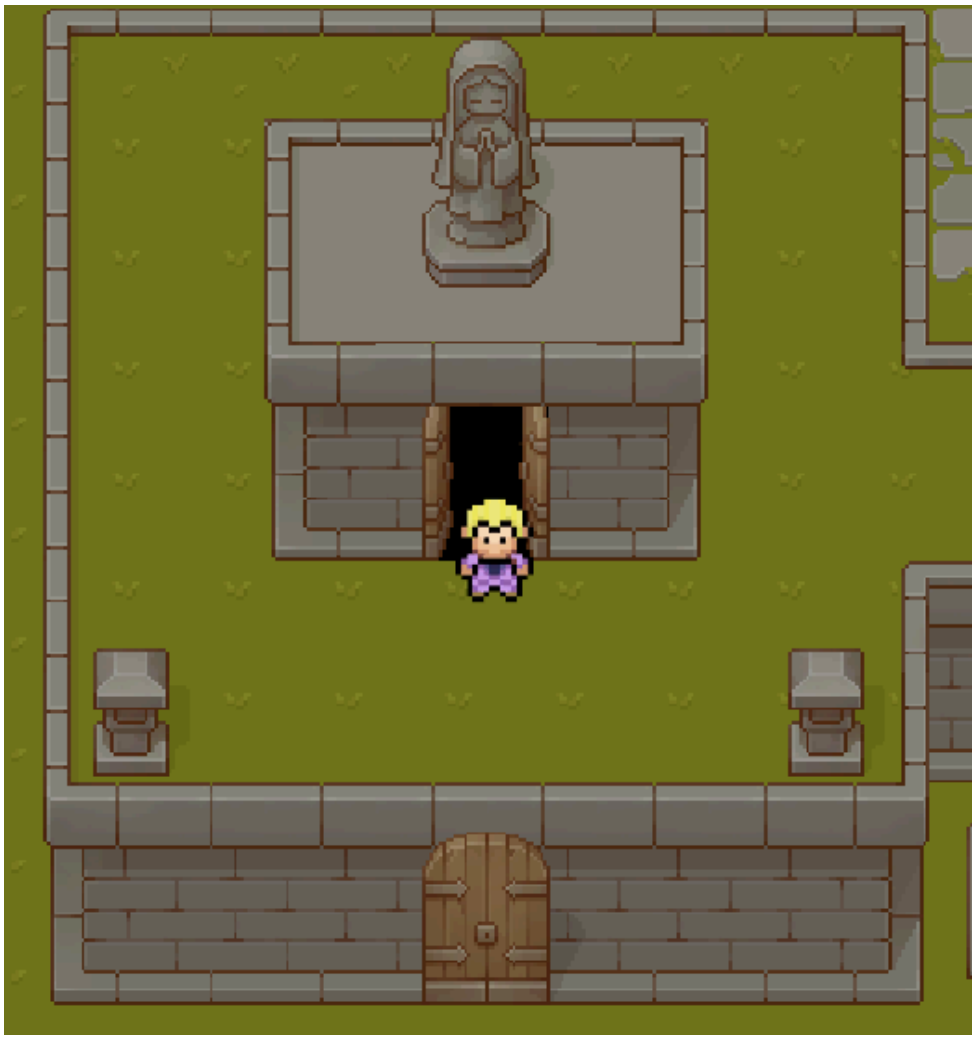

Figure 38: Image of the Protagonist outside a room in the Hub World containing a magic portal.

Once the portals have opened in the Hub World the Protagonist can experience:

- The hostile futuristic **Artificial Intelligence** World.
- The freezing cold **Data Science** World.
- The bright sky islands of the **Cloud** World.
- The scorching heat of the **Threat Intelligence** World.

Figure 39: Image of the Protagonist inside the AI portal room in the Hub World.

After completing a world, the Protagonist will be rewarded with an item, corresponding to the IBM Skills Build course of the world, which can be user to activate a pillar of the altar. Each item offered to the pedestal will strengthen the Spirit of the Altar, prompting it to communicate with the Protagonist more often.

Whenever the Protagonist successfully defeats an IBM co-worker in a World through combat, they then come to their senses and return with the Protagonist to the Hub World, slowly populating the once empty and desolate location. Whilst there, the Protagonist can talk to them to learn more about IBM Skills Build courses. They also tend to give useful advice, so be on the lookout for what they say!

Figure 40: Image of the Protagonist next to an NPC in the Hub World.

After collecting all 4 items, the Spirit of the Altar can use all its powers to open a rift in the world leading to the final world.

5.4.2. Artificial Intelligence World

The AI world takes place in what seems to be an active battlefield. Whether it be the near constant raining of bullets from turrets, or the head seeking explosive drones ready to chase down anything that moves, or the active minefields that will punish any intruder's missteps, nobody can make even a single mistake in this environment.

But fret not, with your guidance and the helpful magic of the Spirit of the Altar (which miraculously grants anyone with incredible durability), the Protagonist will be able to push through till the end!

5.4.2.1. Heart System

With the Altar's magic, the Protagonist is able to take quite a few hits. After each time damaged, they are able to ward off anymore threats for a good two seconds, and only after touching three deadly projectiles would they faint.

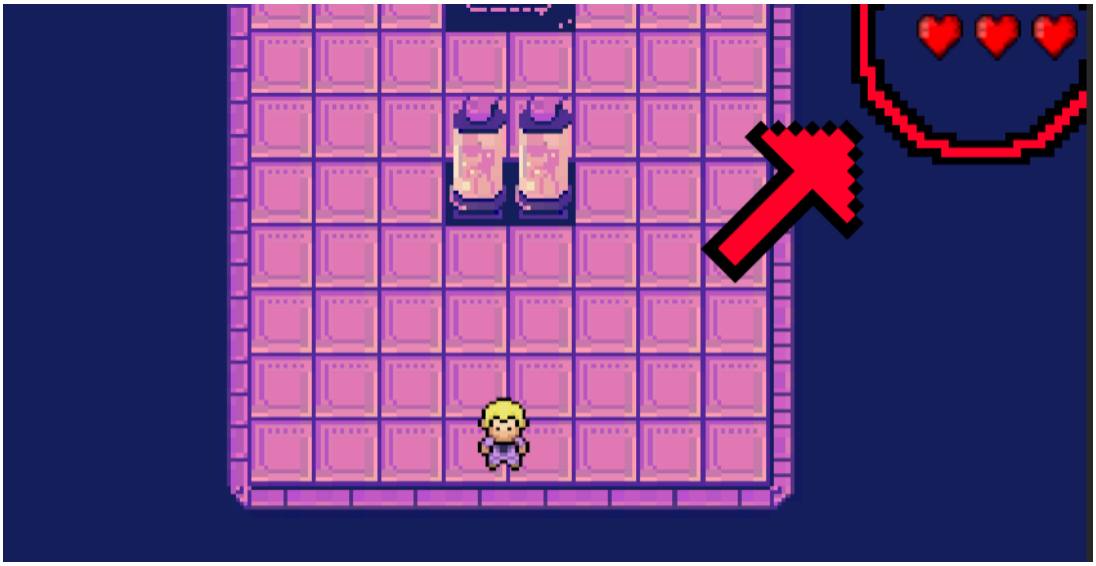

Figure 41: Image showing the heart system, has can be seen the player starts with 3 hearts.

The Altar has also set up a spell on the Protagonist's body that will teleport them to a safe place once they lose consciousness.

5.4.2.2. Projectiles

The Protagonist might not have noticed this, but there seems to be three types of different projectiles commonly used here, which are:

1. **Bullets:** These have a set speed and constantly moves in one direction.
2. **Beams:** These travel at the speed of light, instantly reaching a target, be it a wall or the Protagonist.
3. **Drones:** These are slow moving entities which automatically home towards the Protagonist, exploding when they are in close proximity. These drones are highly experimental, and thus will explode on their own shortly after being released.

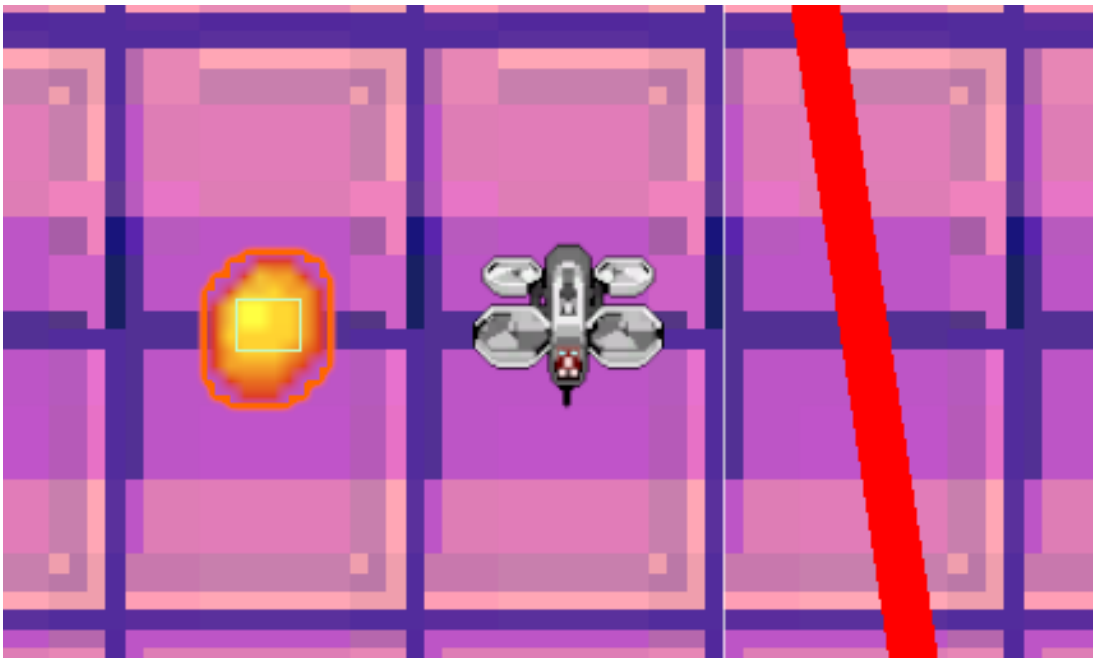

Figure 42: Image showing the 3 types of projectiles side by side with the left being the bullet, the middle one being the drone and the left one being the beam.

By identifying the different threats, you can devise specific strategies that will help the Protagonist deal with the threats individually and avoid the jaws of death.

5.4.2.3. Turrets

The projectiles are shot by sentient turrets which have gone rogue. There are four broad types of turret modes:

1. **Stationary:** The default factory setting, where the turrets stand still facing one direction, and shoot at a constant fire rate.
2. **Rotating:** An advanced setting, promising 360° coverage about a fixed point.
3. **Mobile:** - A setting which toggles the wheels on the turrets, allowing it to move (and potentially rotate as well) between two fixed points.
4. **Experimental:** - Reserved only for in-house testing, this mode goes well beyond the commercially available functionalities, allowing for phases changes while also increasing the variety of projectiles that can be shot.

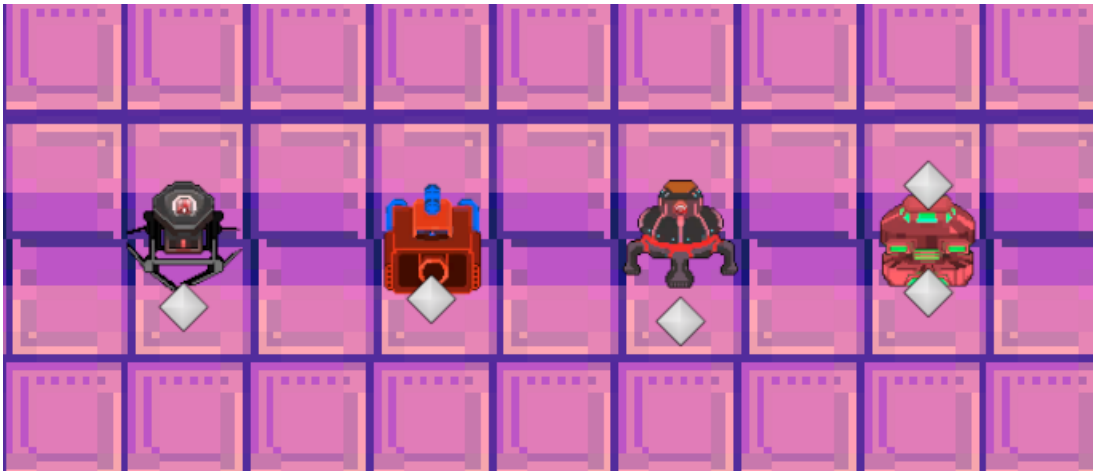

Figure 43: Image showing a stationary turret, a rotating turret, a mobile turret and a experimental turret (from left to right).

5.4.2.4. Abilities

The Protagonist gains access to two abilities, which are unlocked throughout the world.

The first is to **dash**, which allows the Protagonist to move fast in the direction they are currently moving for a short distance. After dashing, there is a very short cooldown which disallows the Protagonist from dashing again.

The second is to **shield**, which protects the Protagonist from bullets and beams. Whilst the shield is active, it continuously consumes energy from the built-in battery. When remaining charge reaches 0, the player is unable to shield anymore. Since the battery life on the shield is very poor, you must help the Protagonist carefully ration out the uses, reserving it only for dire situations.

5.4.2.5. Pickups

There are a few types of items lying around on the ground in this place, presumably left there on purpose.

5.4.2.5.1. Hearts

These are beneficial pickups which will heal up the Protagonist, letting them survive one more hit. The maximum amount of hearts that can be held by the Protagonist at all times is five, since this is the extent of the Altar's magic.

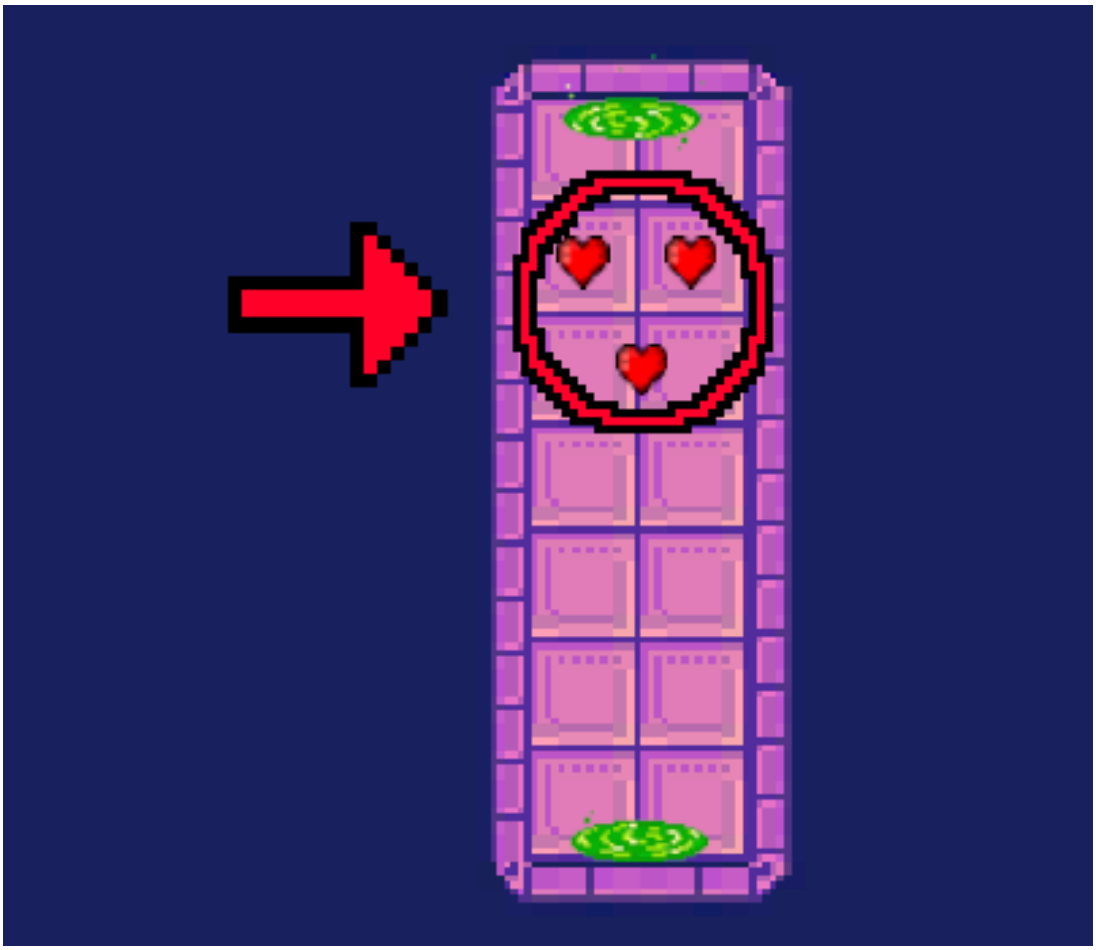

Figure 44: Image showing hearts which can be picked up by the player.

5.4.2.5.2. Keys

In some places, you will see walls blocking the path to progress the level. To get rid of this barricade, you must guide the Protagonist to pick up the green keys on the screen. Only after collecting them all will the path finally be opened.

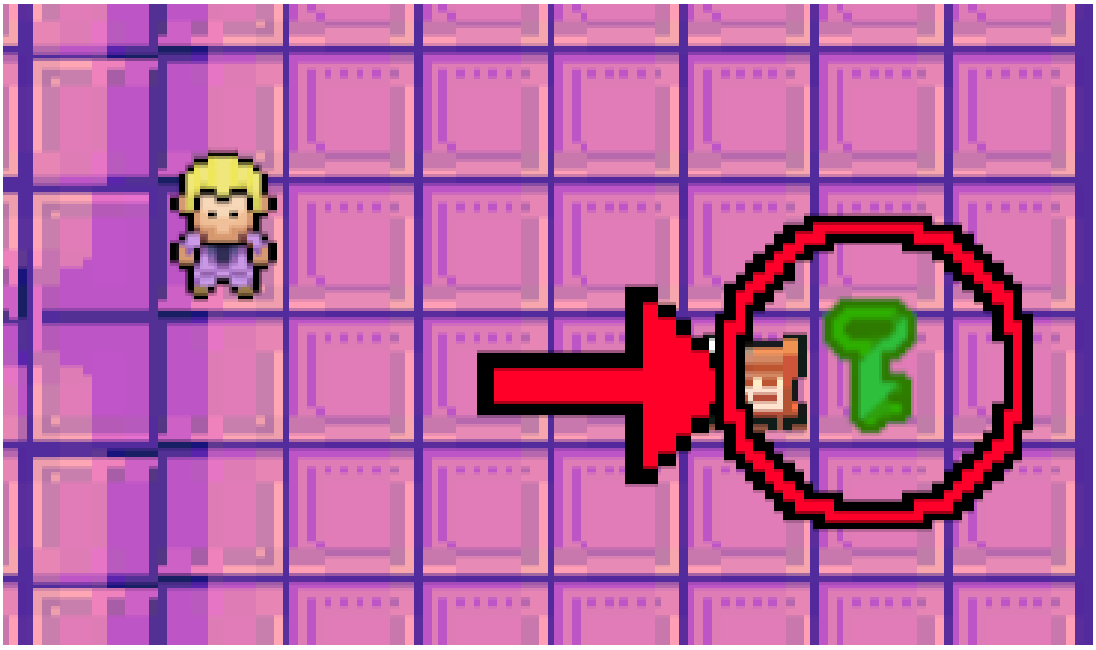

Figure 45: Image showing the keys to collect throughout the level to progress.

5.4.2.5.3. Boots of Speed

A rare item which only shows up once in this world, the Boots of Speed makes the Protagonist much more agile, allowing them to swiftly manoeuvre around the projectiles shot by the turrets. Having permanent access to these would be great, however the boots works only when paired via a Bluetooth with a nearby device, so wearing them to the next level would (unfortunately) not grant the Protagonist extreme speed. Do not be too sad, as even if the shoes are non-functional, they still look great on the Protagonist.

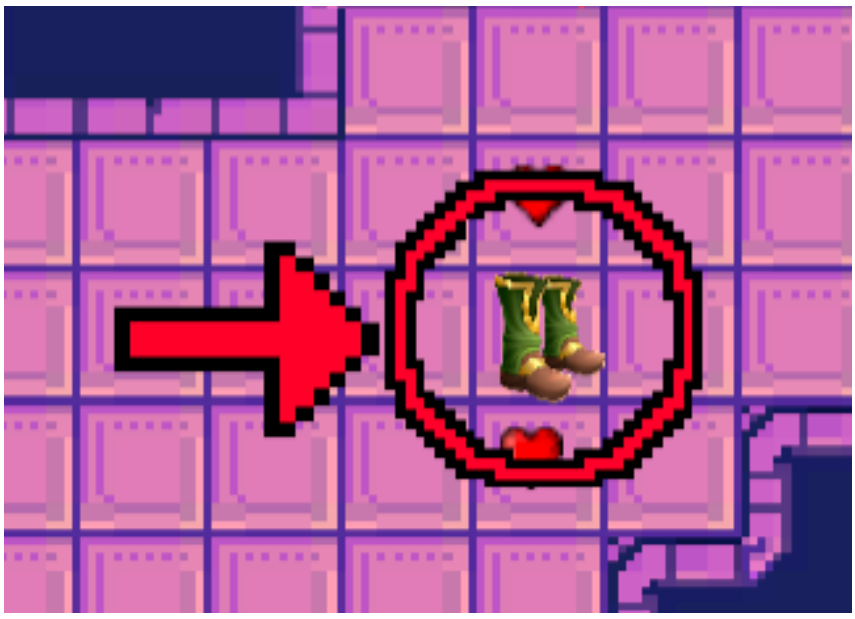

Figure 46: Image showing the Boots of Speed, which increases the player's movement speed.

5.4.2.6. Portals

There are two types of portals present in this world:

1. **One-way:** One-time use gates. These usually appear in at the end of a level and are used as a means of transportation between levels.
2. **Two-way:** Stable portals that are used inside a level. Utilizing these in a clever way will greatly help the Protagonist avoid projectiles! Note however that these have a short cooldown between uses as the portals have to recharge. It also prevents the Protagonist from teleport back and forth indefinitely.

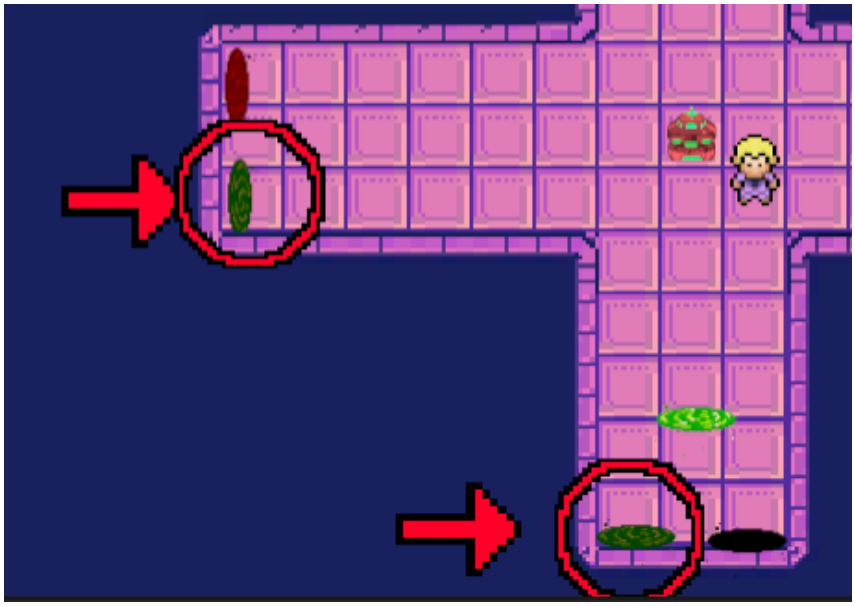

Figure 47: Image showing the two-way portal (in dark green) and one way portal (in light green, above the bottom portal).

5.4.2.7. Moving platforms

In some levels, there are moving platforms which move between 2 points at a constant speed. Moving along with these can help block incoming projectiles. Use these wisely!

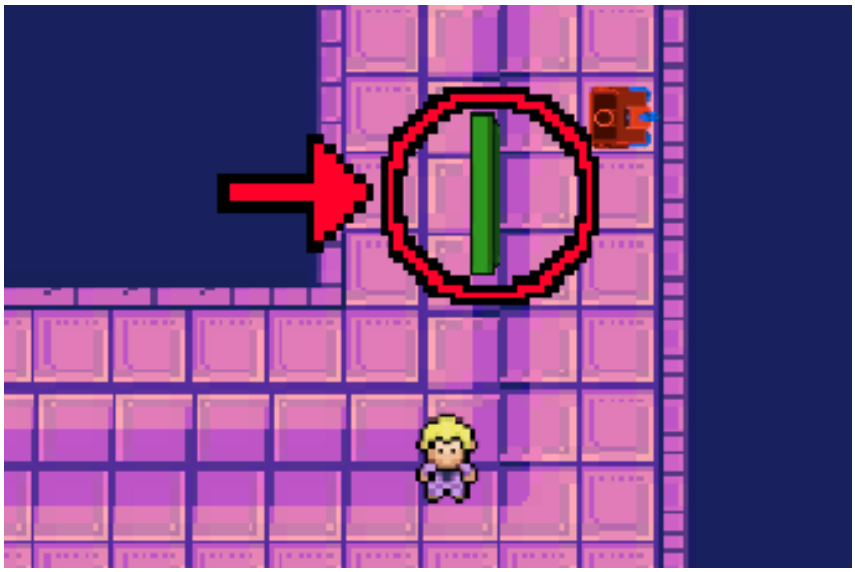

Figure 48: Image showing a moving platform which moves up and down, blocking the turret on the right.

5.4.3. Data Science World

The Data Science world takes place in an icy cave. This cave hides secrets of forgotten life, with the relics of a civilization present, eveident by man made pillars scattered around to hold unstable ice chunks on the roof up, and bookcases containing knowledge of past times. Sometimes the cave is nice to you; with open planes of snowy surfaces and illusory mist doors. But sometimes can be treacherous; with icy floor and heavy snowfall. Sliding on ice is natural, just don't get stuck sliding on it for eturnity! Even the Altar Spirits power can't help you overcome that challenge, so tread carefully!

5.4.3.1. Icy Floor

The DS World contains 6 levels with 4 unique puzzles. A common mechanic present in these puzzles is the ice tiles, which when touched impairs the player's movement for a couple of seconds until they either touch a tile that isn't ice or after 5 seconds are up. The addition of ice tiles makes movement difficult and therefore slows down the player's progress in their task, which is why it is paired with timed puzzles. Can you handle the cold?

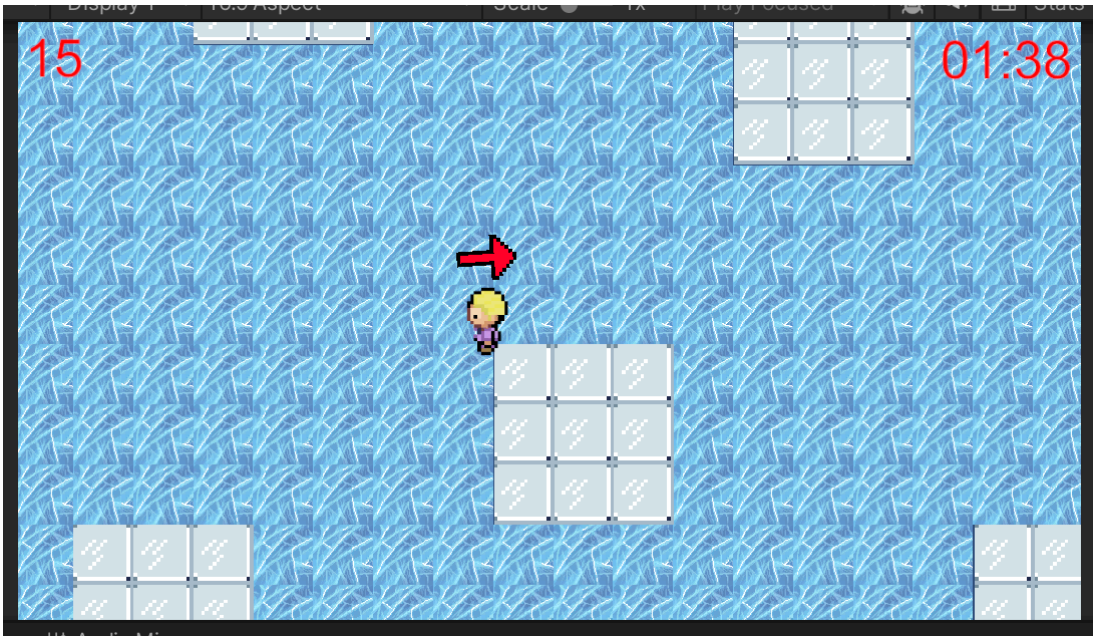

Figure 49: Image of ice tiles in DS World.

5.4.3.2. Maze Doors

In levels 2 and 4 the player can find the Maze Doors puzzle in which they must guess the correct combination of doors to enter to continue. When entering one of the cloudy doors, the user will hear a confirmation sound effect whether the door they entered is correct or incorrect. The player is encouraged to memorize the combination of doors to advance.

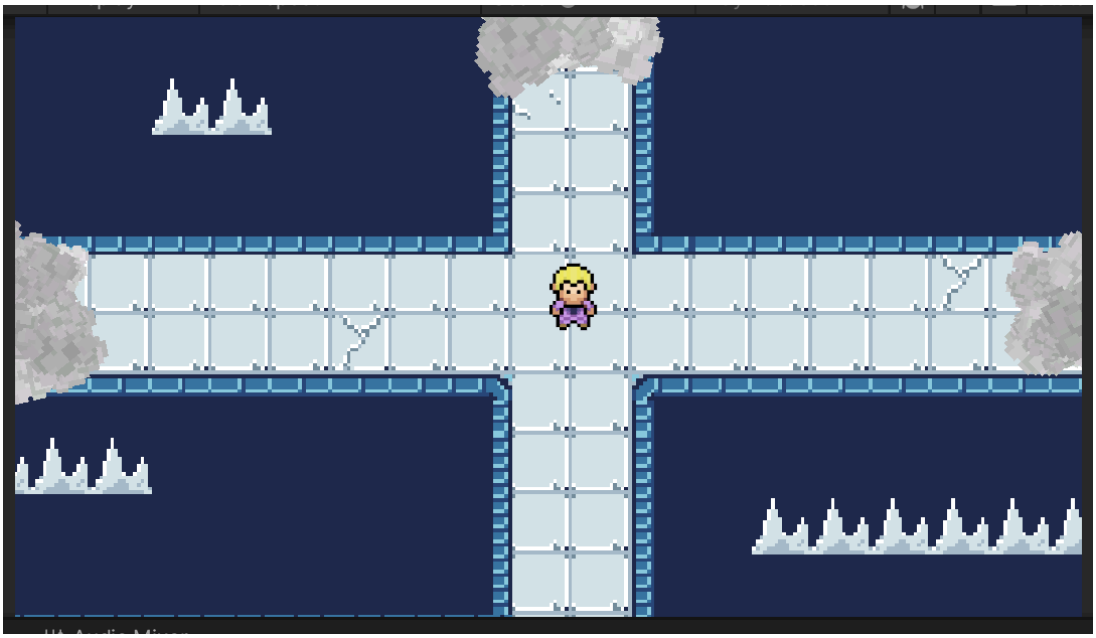

Figure 50: Image of Maze Door Puzzle with 3 exits in DS World Level 2.

5.4.3.3. Timer Challenge

In level 3, the player can find a Timer Challenge, in which they must race from the start line to the finish line within 60 seconds. Failing to do so within the time resets the player back to the beginning of the level.

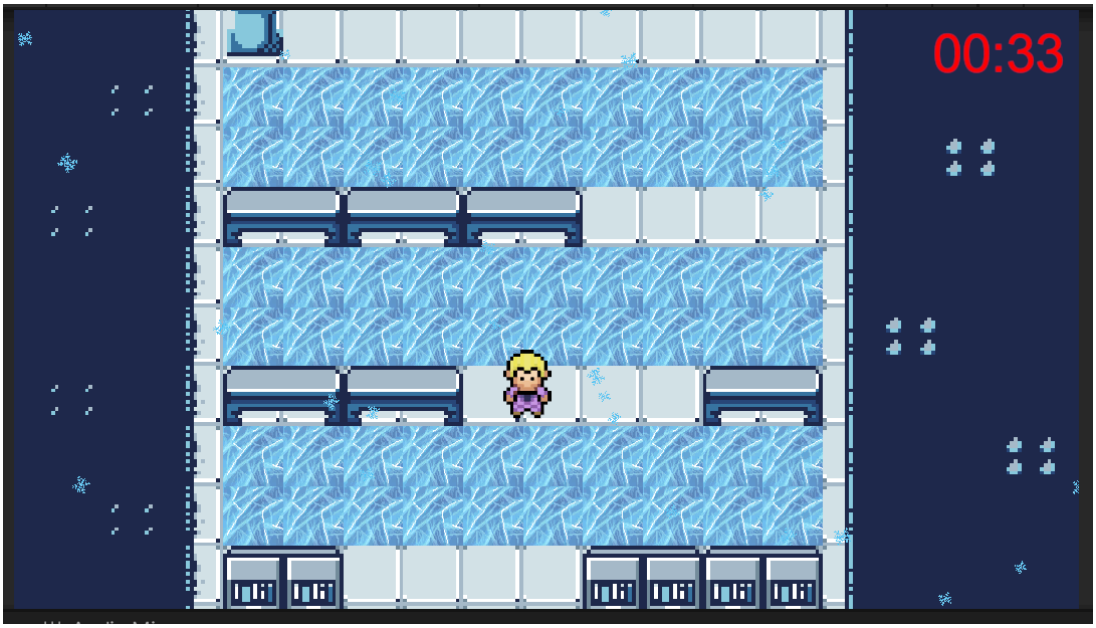

Figure 51: Image of Timer Challenge in DS World Level 3.

5.4.3.4. Button Run

In level 5, the player can find the Button Run, in which the player must hit a button 15 times. The catch is that when the player hits the button, it moves to a different, random location out of a set list of locations. This challenge is timed, and failing to hit the button 15 times within the allotted time results in the level being reset.

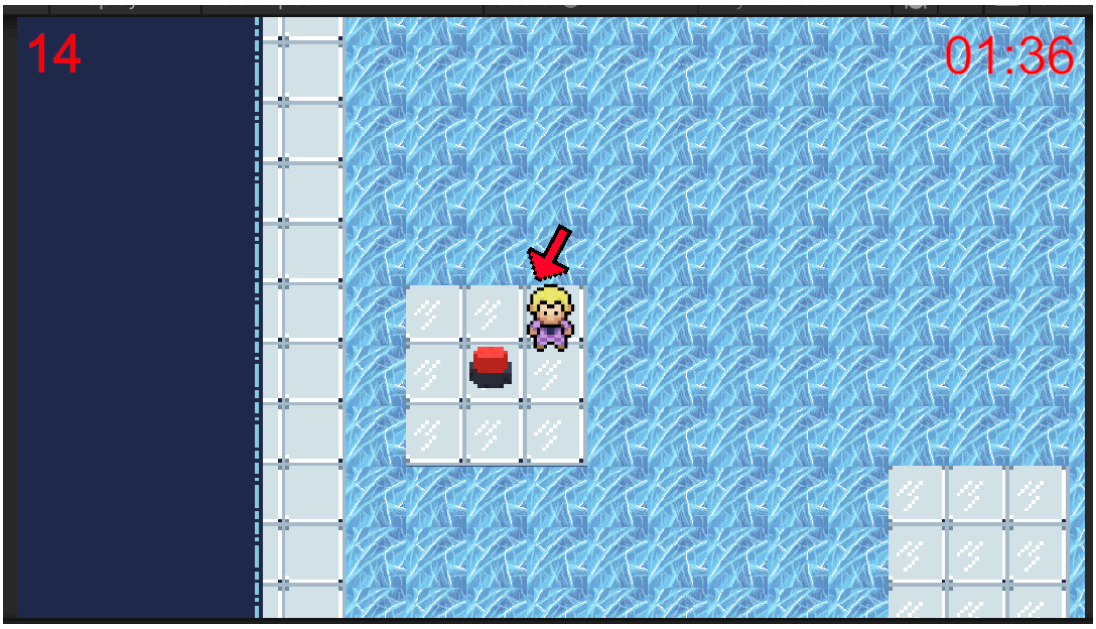

Figure 52: Image of Button Run in DS World Level 5.

5.4.3.5. Dark Maze

In level 6, the player can find the Dark Maze, in which the player must find the center of a maze whilst having a limited vision of their surroundings. This challenge is timed, and failing to find the center of the maze within the allotted time results in the level being reset. Don't get lost!

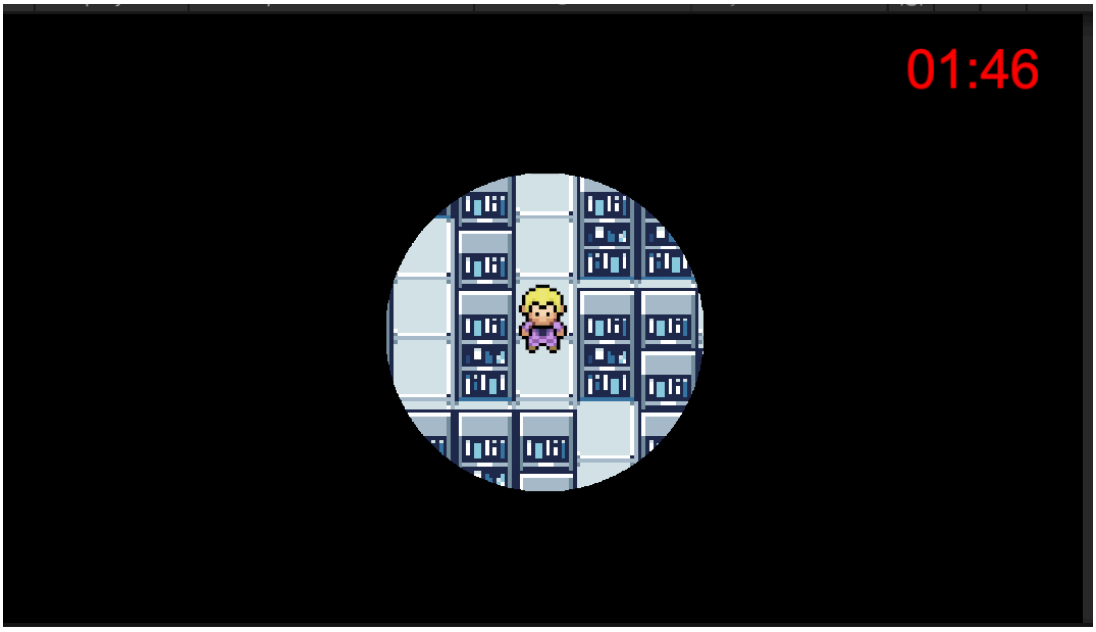

Figure 53: Image of Dark Maze in DS World Level 6.

5.5. Combat

Upon reaching the end of any world, the player is met with a final encounter which will test their understanding of the concepts learnt. These enemies are corrupted IBM co-workers who are masters in their respective fields, and only by proving yourself to be knowledgable and worthy would they come back to their senses and hand over the relics required to face the creator of this world!

Figure 54: Image of a corrupted employee conversing with the player character.

5.5.1. Combat Sequence

Following the discussion with the co-worker, you will be shown a brief cutscene with some intense combat music before the screen fades into the fight sequence. Both the player and enemy combat sprites, alongside their animations, have been manually handcrafted to make the game feel more alive, and the background used for the fights seamlessly matches up with the theme of the world. After all, what good showdown would be complete without an epic title card sequence of the combatants?

Figure 55: Image of the pre-combat cutscene.

During a fight, the co-worker will ask the player a set of questions taken from the current world's Skills Build badge quiz. When answered correctly, they will take damage, whereas if answered incorrectly, the player will take damage. The questions will be asked until one of the two character's health reaches zero. Setting up the Q&A in this manner makes the whole experience more exciting and different compared to the traditional quizzes.

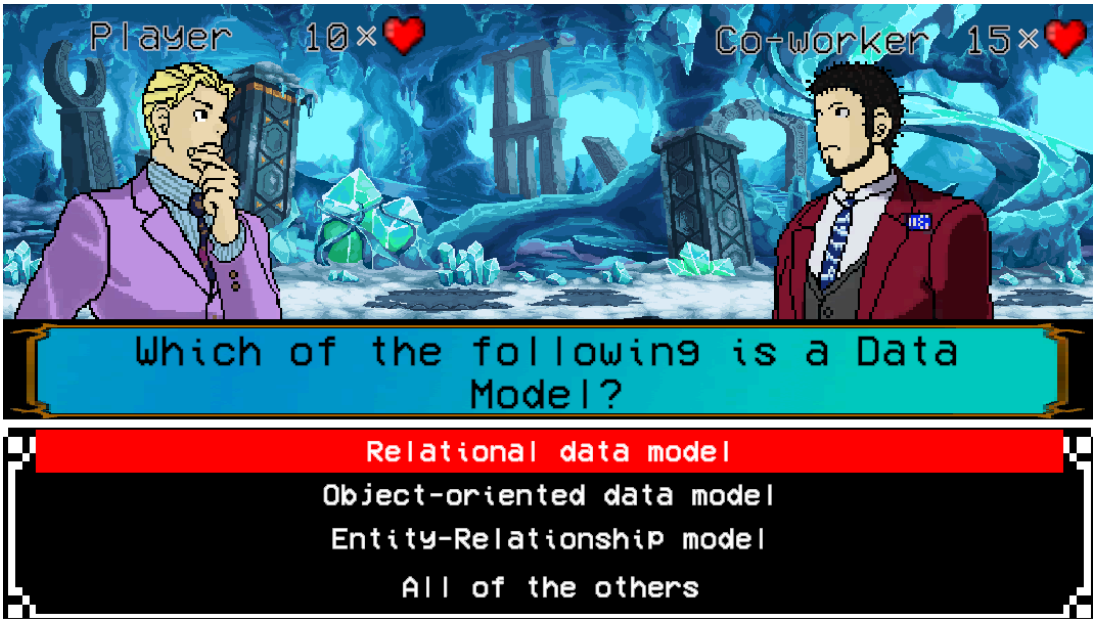

Figure 56: Image of the combat sequence.

Losing the fight sequence will take the player back to the start of the combat stage, allowing them to retry the encounter, whereas winning will bring you to an unfamiliar selection screen.

5.5.2. Skills Build™ System

After defeating a co-worker and helping them snap out of the corruption, the player will receive the relic. These artefacts are so powerful that merely being in the vicinity grants unimaginable powers; no wonder the player's co-workers are so strong! With this, you are able to grant the player one of two special bonuses, permanently enhancing their combat capabilities to make future fights easier.

Figure 57: Image of Skill Tree power-up option after combat.

6. Technical Manual

In this section of the user manual, we will be going over the technical details of our project. We recommend that you go through this section after having a firm grasp of the game, which was covered in Section 5.

6.1. Code Accessibility

To start working on our project and make changes, a few preliminary steps will have to be done.

6.1.1. Getting Project Files

To download our code base, follow these simple steps:

1. **Locate the repo:** To find the repo containing our code base, access the webpage: https://github.com/COMP2281/software-engineering-group-17. As this is a private repository, you will be required to ask COMP2281 for access.
2. **Choosing the branch:** Currently, our code base is located on the "Unity-Game" branch. From the drop-down menu under the project name, which should originally have "main" on it, select "Unity-Game".
3. **Downloading the repo:** Once on this branch, click on the green button that says "Code", and download the code base either as a .zip file, or by cloning it. Note that this does require Git to be installed on your machine.

6.1.2. Setting Up Unity

The entire project was built using Unity, therefore it is crucial to install it correctly. To do so, follow the steps shown below:

1. **Download Unity Hub:** To do so, access the webpage: https://unity.com/download, and choose the downloader appropriate for your OS system.
2. **Install Unity Hub:** Once the download is finished, run the executable file. Follow the instructions until the installation is finished.
3. **Choosing the editor version:** For this project, make sure to install the editor version **2022.3.14f1**. This step is crucial, as using the wrong version of the editor will break everything!
4. **Add project on Unity Hub:** On the project dashboard of Unity Hub, press on the "Add" button. Then, and locate the previously downloaded project code base, go into the "game" folder, select "SEProject" and press open.
5. **Opening the project:** On your dashboard, you should see a new project. Pressing on it will open the project in the Unity Editor. This is where you can see the scenes and change things.

6.1.3. Working with Unity

Working with Unity can be challenging for new users. However, there are lots of resources online that help alleviate this, such as the Unity Editor documentation, the various tutorials on YouTube covering general concepts, as well as a plethora of blog posts which go in-depth on niche problems. Seeing as most of these resources are aimed at younger individuals wanting to be game developers, the content is very light and understandable. For project-specific questions, there are discussions boards on the official Unity website with people willing to help you answer them.

In this section, we will be giving a brief introduction to the Unity Editor, how to work with it, as well as talking about some fundamental concepts.

The Unity Editor is split into 6 views: Project, Scene, Game, Console, Hierarchy, and Inspect.

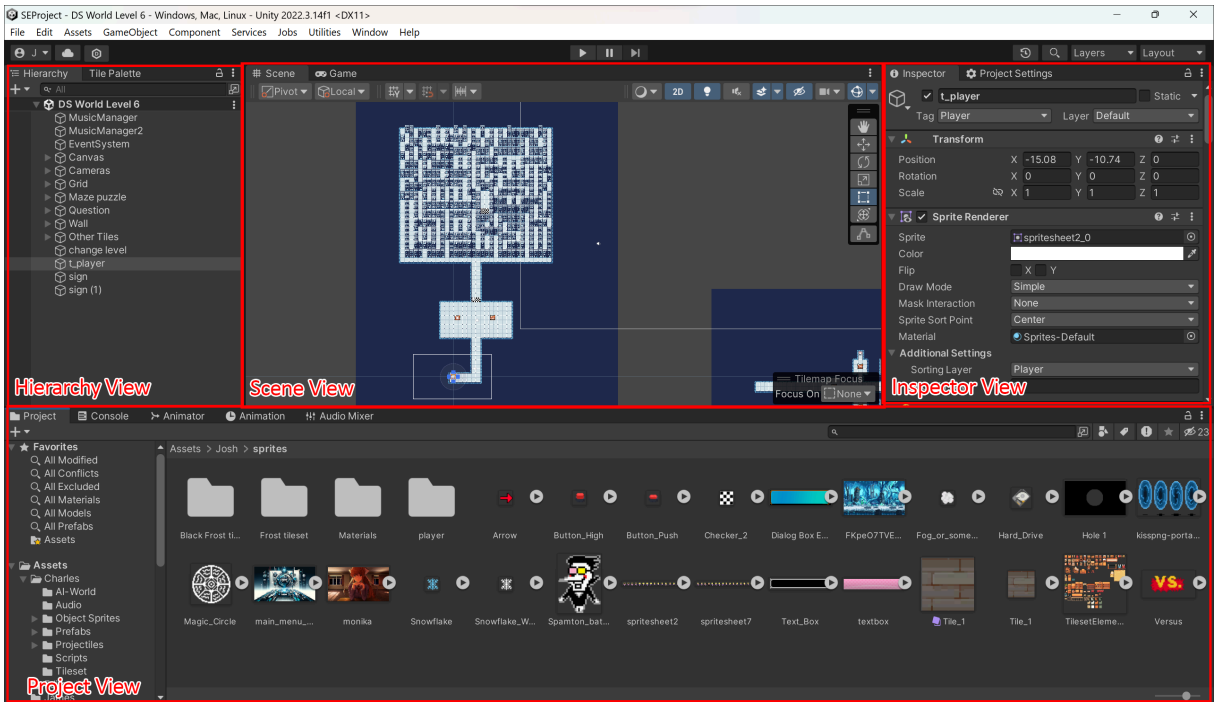

Figure 58: Image of the Unity Editor split into the 4 main views. The game view is another tab inside the Scene view window, and the Console is another tab inside Project view.

- **The Project view** is the main folder which holds all of your game files, such as scenes, objects, prefabs and much more. Think of this as a root folder for the code base.
- **The Scene view** is an interactive editor for individual scenes. A scene can be thought of as a canvas developers can add objects onto. This is one of the main parts that will be used to design the game, with everything in the scene being viewable by the user when playing the game. In this view, objects can be moved around the scene, rotated, scaled and transformed.
- **The Game view** is what the user sees when they play the game. This is essential for testing whether everything in the scene works or not. To build the scene into the Game view, press the play icon up top in the center of the screen. While in this view, you can pause the game, change public variables' values, and even move objects in the scene around.
- **The Console (view)** acts like any other consoles in an IDE. Whenever an error is thrown in the game, it will be shown in the console. You can also use the console to test your code with print statements.
- **The Hierarchy view** is where all objects in the Scene view can be found, broken down to its components. For example, a dialogue box that you see in the Scene view could be made up of a background image and a textUI component inside the Hierarchy. The Hierarchy can be used to quickly select an item in the scene, even when they are invisible.
- **The Inspect view** shows different attributes that make up an object component selected in the Hierarchy. Not only that, it allows developers from accessing an object's public variables and allows scripts to be attached to objects. This is the main tool for adding functionality to the scene objects, such as collision for walls, or health for a player character.

The following is the general workflow in Unity.

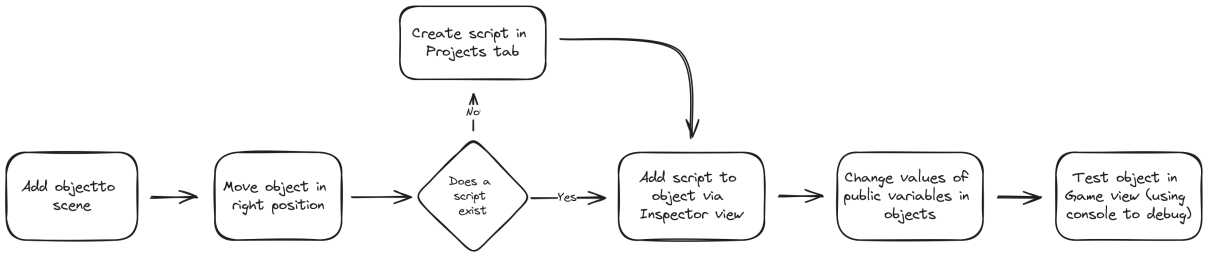

Figure 59: Image of the Unity workflow.

6.1.4. Integrated Development Environment

As our code base is written in C#, you are not forced into using any specific IDEs, as long as you feel comfortable with it. However, we recommend using either VSCode with the C# extension, or using Visual Studio as it is the default IDE for a Unity project. Visual Studio also includes some built-in functionality that might be useful, like a UML diagram generator.

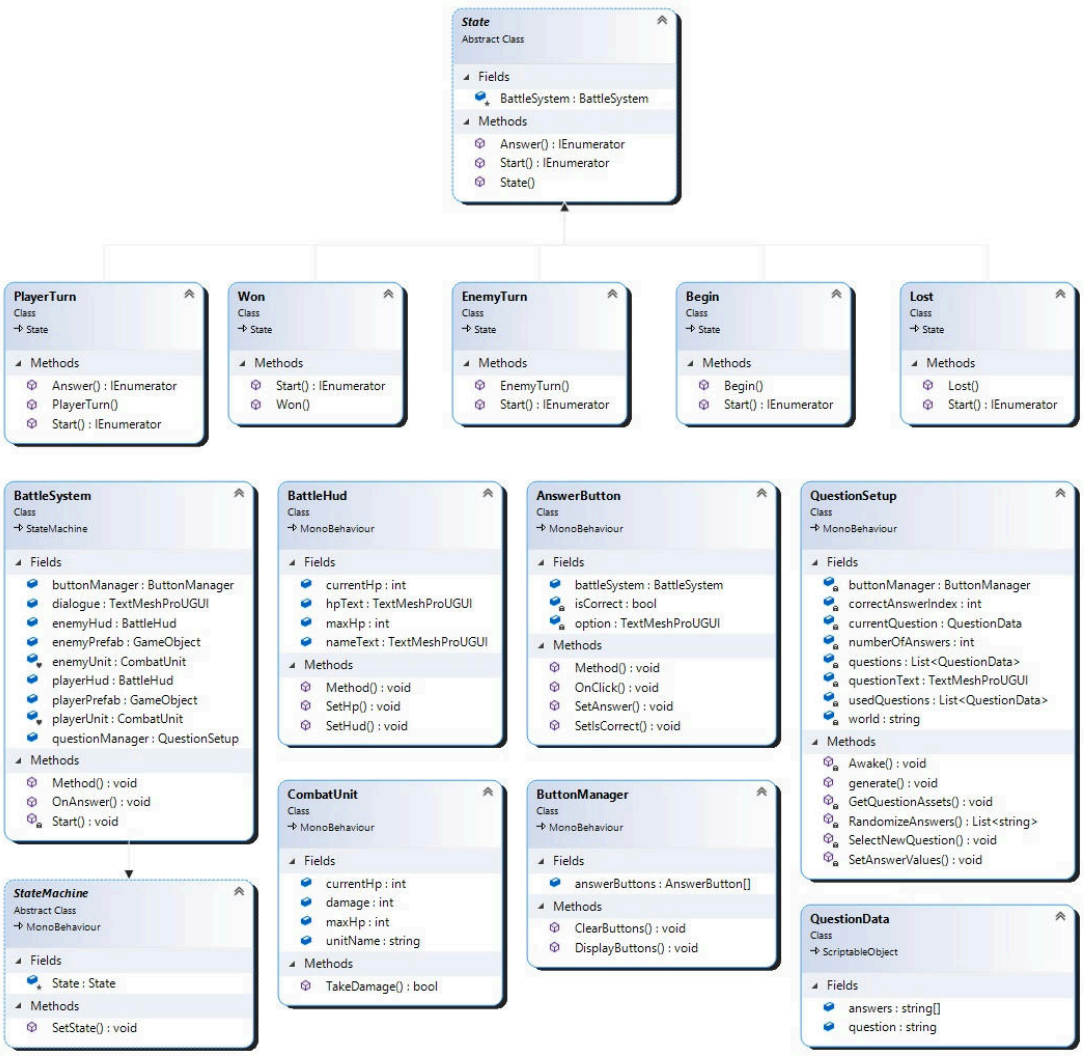

Figure 60: Image of the combat system UML diagram.

6.2. Class documentation

In this section, we provide the documentation for the classes used to build the game. The following section will be divided into general classes which are used throughout the game, as well as world-specific classes.

6.2.1. Overview

The following section is split into two parts. The first part is a table overview of the classes, containing its name, description and functions. The second part is an extensive description of what each class' functions do, and includes images to show where one can find the classes used in.

6.2.1.1. General Classes

Class	Description	Functions
`AnswerButton` (Section 6.2.2.1)	Handles functionality for combat answer buttons	`SetAnswer` `SetIsCorrect` `OnClick`
`Arrow` (Section 6.2.2.2)	Point an arrow `GameObject` towards a specified `GameObject`	`Update`
`BattleCutscene` (Section 6.2.2.3)	Displays the cutscene before combat	`Start` `Update` `FadeOutRoutine`
`BattleHud` (Section 6.2.2.4)	Handles combat HUD information display	`SetHud` `SetHP`
`BattleSystem` (Section 6.2.2.5)	Handles combat using a finite state machine system. Inherits from `StateMachine`	`Start` `OnAnswer`
`Begin` (Section 6.2.2.6)	One of the combat states. Responsible for the start sequence of the fight. Inherits from State	`Start`
`ButtonManager` (Section 6.2.2.7)	Handles functionality which affects all the combat buttons at once	`DisplayButtons` `ClearButtons`
`CombatUnit` (Section 6.2.2.8)	Stores key information about either the player or the boss for fights, including `unitName`, `damage`, `maxHP`, and `currentHp`	`TakeDamage`

`CoWorkerCutscene` (Section 6.2.2.9)	Display the cutscene when talking to a co-worker	`Update` `OnTriggerEnter2D`
`CoWorkerSpawner` (Section 6.2.2.10)	Spawns Co-Workers in the Hub World when they have been defeated in combat	`Update`
`Credits` (Section 6.2.2.11)	Play the end credit scene	`Start` `Update` `TheEnd`
`EnableArrow` (Section 6.2.2.12)	Enables the arrow `GameObject`	`OnTriggerEnter2D`
`EnemyTurn` (Section 6.2.2.13)	One of the combat states. Responsible for the enemy's turn. Inherits from State	`Start`
`GM` (Section 6.2.2.14)	Defines utility methods that allow other classes to manipulate game information such as game save data and game volume	`Update` `Awake` `GetParticle` `SetParticle` `GetVolume` `SetVolume` `GetGraphics` `SetGraphics`
`ImportQuestions` (Section 6.2.2.15)	One of the combat states. Responsible for the enemy's turn. Inherits from State	`Start`
`Inventory` (Section 6.2.2.16)	Loads correct sprites in the inventory.	`Update`
`Lost` (Section 6.2.2.17)	One of the combat states. Responsible for handling what happens after the player loses a fight. Inherits from State	`Start`
`MainMenuBackground` (Section 6.2.2.18)	Moves the background image around at random intervals and blurs the background image at random intervals	`Start` `Update`
`MainMenuButtons` (Section 6.2.2.19)	Implements the functionality for the game's Main Menu. These include starting the game, entering full-	`Start` `NewGame` `Quit` `Volume` `Particles`

	screen mode and modifying the game's volume.	`FullScreen`
		`MainMenu`
		`OptionsMenu`
		`SetQuality`
		`Credits`

`MusicManager` (Section 6.2.2.20)	Manages the music in all scenes	`Awake`
		`PlayMusic`
		`StopMusic`
		`ChangeMusic`

`PauseMenu` (Section 6.2.2.21)	Defines methods related to the game's Pause Menu. Implemented functionality includes pausing the game, resuming the game and modifying the game's volume	`Start`
		`Resume`
		`Pause`
		`QuitGame`
		`MainMenu`
		`OptionsMenu`
		`Volume`
		`Particles`
		`SkillsBuild`
		`SkipLevel`
		`DisplayText`
		`SetQuality`

`PlayerTurn` (Section 6.2.2.22)	One of the combat states. Responsible for the player's turn. Inherits from State	`Answer`
`PostBattleCutscene` (Section 6.2.2.23)	Responsible for loading the cutscene after a Boss fight	`Update`, `OnTriggerEnter2D`
`PowerUp` (Section 6.2.2.24)	Responsible for increasing the players stats after combat	`Attack`, `Hp`
`PreBattleCutscene` (Section 6.2.2.25)	Responsible for loading the cutscene before the Boss fight	`Update`, `OnTriggerEnter2D`
`QuestionData` (Section 6.2.2.26)	Stores the information necessary for a question	None

`QuestionSetup` (Section 6.2.2.27)	Generates a question for combat. The order in which they are generated is randomized, and if all questions are exhausted then allow for reuse	`Awake`
		`GetQuestionAssets`
		`Generate`
		`SelectNewQuestion`
		`SetAnswerValues`
		`RandomizeAnswers`

`Sign` (Section 6.2.2.28)	Used to display dialogue when `GameObject` is interacted with	`Update` `OnTriggerEnter2D` `OnTriggerExit2D`
`State` (Section 6.2.2.29)	An abstract class which represents a state in the combat sequence. Every combat state inherits from this. initialises with `BattleSystem`	`Start` `Answer`
`StateMachine` (Section 6.2.2.30)	An abstract class which represents a finite state machine. `BattleSystem` inherits from this	`SetState`
`Won` (Section 6.2.2.31)	One of the combat states. Responsible for handling what happens after the player wins a fight. Inherits from State	`Start`

6.2.1.2. Hub World classes

Class	Description	Functions
`AltarCutscene1` (Section 6.2.2.32)	Loads dialogue for the first cutscene with altar	`Update` `OnTriggerEnter2D`

`AltarCutscene2` (Section 6.2.2.33)	Loads dialogue for the second cutscene with altar	`Start`
		`Update`
		`OnTriggerEnter2D`

`AltarCutscene3` (Section 6.2.2.34)	Loads dialogue for the third cutscene with altar	`Start`
		`Update`
		`OnTriggerEnter2D`

`AltarCutscene4` (Section 6.2.2.35)	Loads dialogue for the fourth cutscene with altar	`Start`
		`Update`
		`OnTriggerEnter2D`

`AltarItemGet` (Section 6.2.2.36)	Loads dialogue for cutscene when the player has got an item	`Update` `OnTriggerEnter2D`
`CreepyCutscene` (Section 6.2.2.37)	Load dialogue boxes of cutscene and open a URL link	`Update`

`HubDoor` (Section 6.2.2.38)	Open and close the door `GameObject` in the Hub World when interacted with	`Update`
		`OnTriggerEnter2D`
		`OnTriggerExit2D`

`HubDoorTrigger` (Section 6.2.2.39)	Move player to a different room when interacted with (used behind a door `GameObject` to simulate entering a room)	`OnTriggerEnter2D`
`HubTeleport` (Section 6.2.2.40)	Load a specified scene when interacted with	`OnTriggerEnter2D`
`IntroCutscene` (Section 6.2.2.41)	Loads dialogue of the introduction cutscene	`Update` `OnTriggerEnter2D`
`GMHub` (Section 6.2.2.42)	Responsible for loading the correct cutscene and coworkers in the Hub World depending on player progression in the game	`Start`
`HubBackground` (Section 6.2.2.43)	Rotate the background image	`Update`

`NamePlayer` (Section 6.2.2.44)	Accesses the player's real-life name. Used during dialogue	`Start`
`Portal` (Section 6.2.2.45)	Script to enable a `GameObjects` sprite and collision	`Start` `Update`

6.2.1.3. Data Science World classes

Class	Description	Functions

`Button` (Section 6.2.2.46)	Button corresponding to the questions on signs, used to input the answer to the signs questions	`Start`
		`Update`
		`OnTriggerEnter2D`
		`OnTriggerExit2D`
		`Return`
		`OpenDoor`

`ButtonSpawner` (Section 6.2.2.47)	Button `GameObject` that when interacted with, moves to a random different location out of a specified list. Must be interacted with a specified amount of times	`Start`
		`Update`
		`Spawn`
		`OnTriggerEnter2D`
		`OnTriggerExit2D`

`CameraFollow` (Section 6.2.2.48)	Used to switch between a camera that follows the player and stationary cameras	`OnTriggerEnter2D`
`CameraManager` (Section 6.2.2.49)	Switch between Stationary cameras	`OnTriggerEnter2D` `OnTriggerExit2D`
`DoorTriggerLeft` (Section 6.2.2.50)	Check door collision. Used in the `LabyrinthDoor` script	`OnTriggerEnter2D`
`DoorTriggerRight` (Section 6.2.2.51)	Check door collision. Used in the `LabyrinthDoor` script	`OnTriggerEnter2D`
`DoorTriggerTop` (Section 6.2.2.52)	Check door collision. Used in the `LabyrinthDoor` script	`OnTriggerEnter2D`
`Hole` (Section 6.2.2.53)	Enables the hole `GameObject`	`OnTriggerEnter2D`
`LabyrinthDoor` (Section 6.2.2.54)	Keeps track of, and updates player progress in a specific puzzle on collision (Puzzle archetype 1)	`OnTriggerEnter2D`
`LevelChanger` (Section 6.2.2.55)	Changes the level to the next level	`OnTriggerEnter2D`

`PlayerIceScript` (Section 6.2.2.56)	`Player` script, but updated to change the ice physics. In this script there is no failsafe for if the player gets stuck on ice	`Update`
		`FixedUpdate`
		`Flip`

		`OnTriggerEnter2D` `OnTriggerExit2D`

`PlayerScript`
(Section 6.2.2.57)
	Enables player movement input	`Update`
		`FixedUpdate`
		`Flip`
		`OnTriggerEnter2D`
		`OnTriggerExit2D`
		`Unfreeze`
		`Frozen`

`RemoveArrow` (Section 6.2.2.58)	Disable the arrow `GameObject`	`OnTriggerEnter2D`

`SecretRoom`
(Section 6.2.2.59)
	If the player is in range of `GameObject`, disable physical collision	`Update`
		`OnTriggerEnter2D`
		`OnTriggerExit2D`

`Snow` (Section 6.2.2.60)	Enables snow particle to fall	`Start` `Update`
`SnowGenerator` (Section 6.2.2.61)	Updates the snow generator's location based on the currently active camera	`Update`

`Spawner`
(Section 6.2.2.62)
	Places the player at a specific location in a scene. Linked to Puzzle 1	`Start`
		`Update`
		`FixedUpdate`
		`OnTriggerEnter2D`
		`MovePlayer`

`SpecialTimer`
(Section 6.2.2.63)
	Starts a timer and counts down to 0. The same as the `Timer` script, only it takes in a different player script	`Update`
		`UpdateTimer`
		`TimerEnded`
		`OnTriggerEnter2D`
		`Restart`
		`PlayMusic`

`Teleporter` (Section 6.2.2.64)	Move the player to a specified location in the scene and disable the hole `GameObject`	`OnTriggerEnter2D`

`Timer`
(Section 6.2.2.65)
	Starts a timer and counts down to 0	`Update`
		`UpdateTimer`
		`TimerEnded`
		`OnTriggerEnter2D`
		`Restart`
		`PlayMusic`

`TimerStop` (Section 6.2.2.66)	Stops the timer from running	`OnTriggerEnter2D`
`TimerStopSpecial` (Section 6.2.2.67)	Stops the timer from running. The same script as TimerStop, only it uses SpecialTimer instead of Timer	`OnTriggerEnter2D`
`WallMaker` (Section 6.2.2.68)	Enables a collision `GameObject` when interacted with	`OnTriggerEnter2D`

6.2.1.4. AI World classes

Class	Descriptions	Functions

`AddEscape` (Section 6.2.2.69)	Adds objects specified and removes other objects specified	`Start`
		`Update`
		`StartEscape`

`AIPortal` (Section 6.2.2.70)	Teleports player on collision to another portal (this portal is specified by the parent object of the portal itself)	`Start` `OnTriggerEnter2d`
`AudioManager` (Section 6.2.2.71)	Manage the music that plays on collision. Attached to an empty object with a collision box for when the player enters/exits	`OnTriggerEnter2D` `OnTriggerExit2D`

`BasicEnemyTurret` (Section 6.2.2.72)	Inherits from the abstract class `TurretStateMachine`, which changes between 4 states	`Start`
		`Update`
		`FixedUpdate`
		`ShootHandler`
		`StartTimer`
		`GetVelocity`
		`Fire`

`Beam` (Section 6.2.2.73)	Draws a beam which damages the player if the laser hits the player, the laser stops at any collision (unless it is within a specific `ignoreRayCast` layer)	`Start`
		`Update`
		`DeathTimer`
		`Shoot`
		`DrawBeam`
		`ManagePhase`

`Bullet` (Section 6.2.2.74)	Moves the attached `GameObject` in its current direction at a specified velocity and damages the player on contact (if the player is vulnerable)	`Update` `OnTriggerEnterCollision2D`
`Dash` (Section 6.2.2.75)	Adds the ability to dash to the player	`Update` `StartDash`

`DashItem` (Section 6.2.2.76)	Grants the player the ability to dash upon collision with the item with this script enabled	`Update`
		`OnTriggerEnter2D`
		`PlayAudio`

`EscapeButton` (Section 6.2.2.77)	Changes the `keyManager`'s `escapeStart` to true to when the player collides with the `GameObject`	`OnTriggerEnter2D`
`Hearts` (Section 6.2.2.78)	Ensures that the number of hearts showing on the canvas corresponds to the number of lives the player has	Start Update ShowHearts

`HomingDrone` (Section 6.2.2.79)	Moves the attached object towards the player and damages the player if too close to the player	`Start`
		`StartPhaseTimer`
		`DeathTimer`
		`FixedUpdate`
		`Update`
		`OnTriggerEnter2D`
		`EndDrone`

`KeyManager` (Section 6.2.2.80)	Activates specified objects and deactivates other specified objects when the number of keys is 0	`Update` `StartMusic`
`Keys` (Section 6.2.2.81)	Reduces the `keyNum` variable in `keyManager` on collision with the player.	`OnTriggerEnter2D`
`KillPlayer` (Section 6.2.2.82)	Damages the player on collision with the player	`OnTriggerEnter2D`

`Lives` (Section 6.2.2.83)	Manages the number of lives of the player, when the player should die, and invincibility	`Start`
		`Update`
		`DamagePlayer`
		`Flicker`
		`InvincibleTimer`
		`OnTriggerEnter2D`

`MovingPlatform` (Section 6.2.2.84)	Moves the object between the two specified `GameObjects` at a constant speed with the specified direction	`Start` `FixedUpdate`
`PlayerMovement2` (Section 6.2.2.85)	Same as the `Player` Script but allows the player to dash	

`PortalParent` (Section 6.2.2.86)	Manages multiple portals and controls what portals the portals teleport to, also manages the cooldown of the portals.	`Start`
		`GetNextPortal`
		`StartCooldown`
		`DisableAll`

`Shield` (Section 6.2.2.87)	Manages the shield which follows the player and blocks projectiles when activated but has a limited amount of time to use	`Start`
		`Update`
		`FixedUpdate`
		`OnTriggerEnter2D`

`ShieldItem` (Section 6.2.2.88)	Activates the shield upon collision with the player.	`OnTriggerEnter2D`
`SpeedBoost` (Section 6.2.2.89)	Increases the player's movement speed upon collision with the player.	`OnTriggerEnter2D`

`TimedRoom` (Section 6.2.2.90)	Starts a timer upon collision with the player and activates a portal after this timer has elapsed	`Start`
		`FixedUpdate`
		`OnTriggerEnter2D`

`Timer2` (Section 6.2.2.91)	Same as the Timer class but only uses 1 audio source	`Update`
		`UpdateTimer`
		`TimerEnded`
		`OnTriggerEnter2D`
		`Restart`
		`PlayMusic`

`TimerStop2` (Section 6.2.2.92)	Stops the timer on collision with player	`OnTriggerEnter2D`

`Turret` (Section 6.2.2.93)	Manages a turret which fires bullets and can rotate	`Start`
		`FixedUpdate`
		`FireBullets`
		`Fire`

`TurretStateMachine` (Section 6.2.2.94)	Defines an abstract class for turrets which have 4 set phases	`SetStartPoint`
		`GetSpawnPoints`
		`MakeEnemy`
		`ShootScatter`
		`ShootStraight`
		`ShootBeam`
		`RotateTurret`
		`GetPrefab`

6.2.2. Detailed Class Description

6.2.2.1. `AnswerButton`

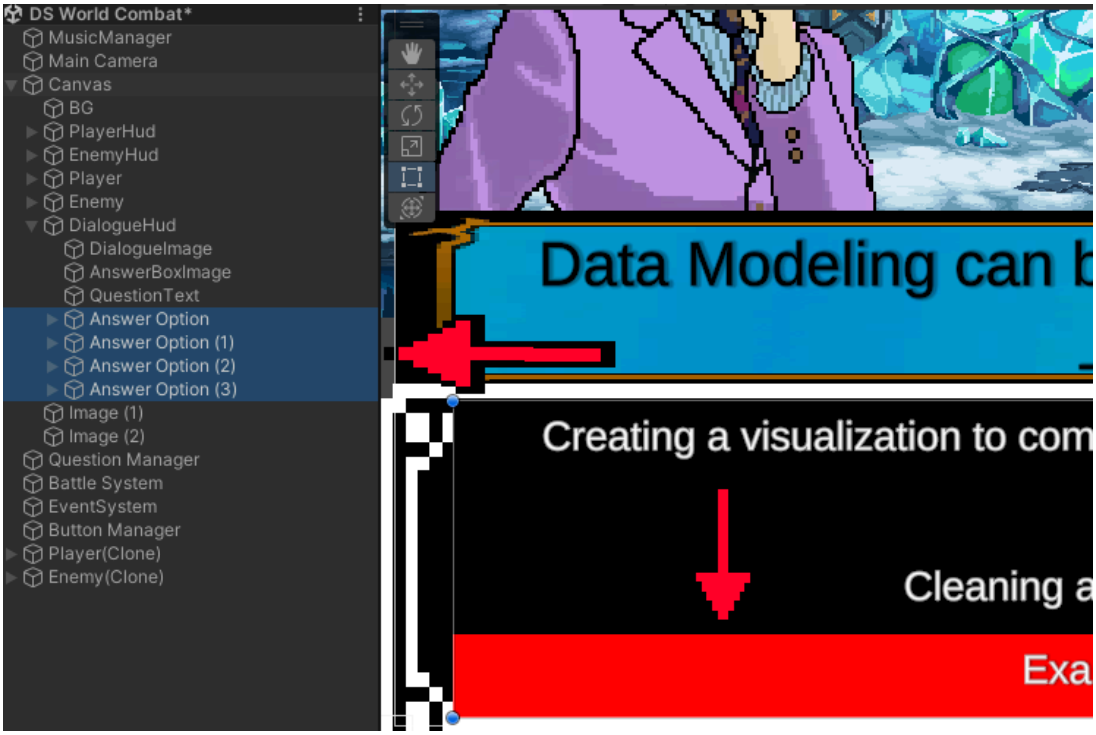

Figure 61: Image showing the `GameObjects` with the `AnswerButton` script attached. These are present in every combat scene. The red box shown is a button that is currently being selected.

Function description

- `SetAnswer`: Set the `option` Text GUI string to the input string.
- `SetIsCorrect`: Set the `newBool` boolean to true or false based on the input.
- `OnClick`: Calls the `battleSystem` `GameObject`'s `OnAnswer` function while passing in either true or false based on `AnswerButton`'s `isCorrect`.

6.2.2.2. `Arrow`

Figure 62: Image showing the `Arrow` `GameObject`, which is linked to the player in Hub World, DS World Level 3 and DS World Level 5 scenes and follows the player when they move.

Function description

- `Update`: Change the rotation of the arrow to always point towards the specified `GameObject`.

6.2.2.3. `BattleCutscene`

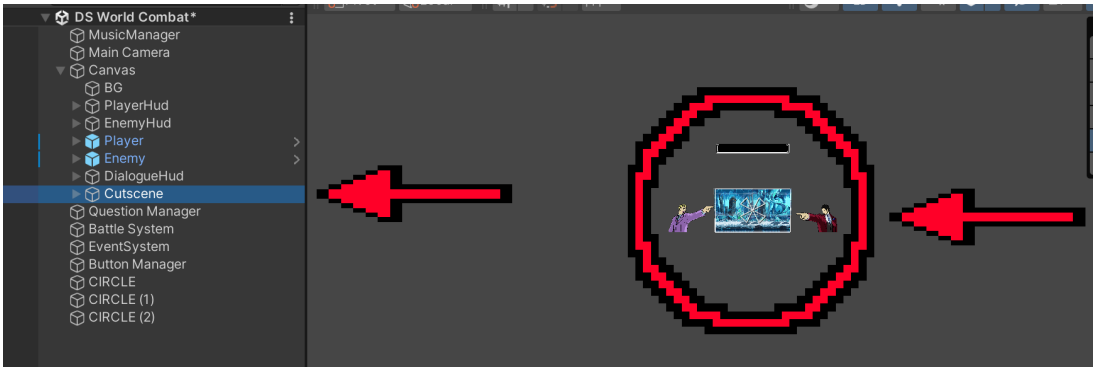

Figure 63: Image showing the `GameObject` with the `BattleCutscene` script attached. This is present in every combat scene and covers the whole canvas.

Function description

- `Start`: Gets all `image components` from every `GameObject` in the cutscene.
- `Update`: Moves all images to their specified location in the `scene`
- `FadeOutRoutine`: Fades out all images in the cutscene.

6.2.2.4. `BattleHud`

Figure 64: Image showing the `GameObjects` with the `BattleHud` script attached. These are present in every combat scene, and can be found on top of the player and enemy character image.

The `BattleHud` script can be found attached to both PlayerHud and EnemyHud, which will be present in every combat scene.

Function description

- `SetHud`: initialise HUD with associated `CombatUnit`'s attributes. This includes `unitName`, `currentHP`, `maxHP`.
- `SetHp`: Set the HP shown on the HUD to input amount. If the input amount is less than 0, set it to zero.

6.2.2.5. `BattleSystem`

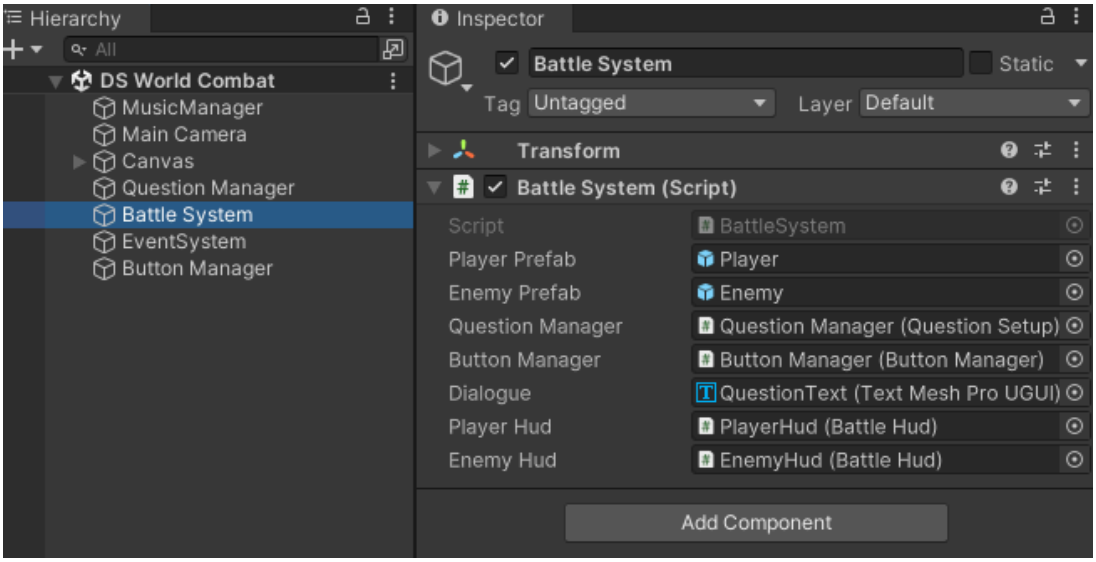

Figure 65: Image showing the `GameObject` with the `BattleSystem` script attached. This is present in every combat scene, but is invisible.

Function description

- `Start`: Call the `SetState` function inherited from `StateMachine`, and pass in a newly created `Begin` state to start the combat sequence.

- `OnAnswer`: Call the `ButtonManager`'s `ClearButtons` function, then run the current state's `Answer` function with a past in boolean from the selected answer.

6.2.2.6. `Begin`

See Section 6.3.3 for more detail.

Function description

- `Start`: initialise the player and enemy object, wait until the combat cutscene finishes playing, then transition into `EnemyTurn` state.

6.2.2.7. `ButtonManager`

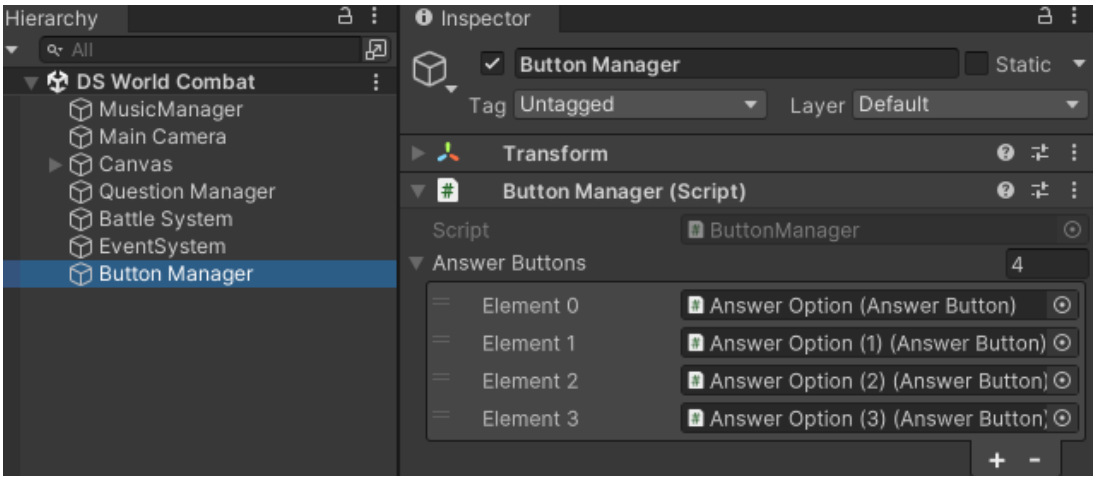

Figure 66: Image showing the `GameObject` with the `ButtonManager` script attached. This is present in every combat scene, but is invisible.

Function description

- `DisplayButtons`: Set a number of buttons to active based on the given input amount. Reset selected to the first button.
- `ClearButtons`: Deactivate all buttons.

6.2.2.8. `CombatUnit`

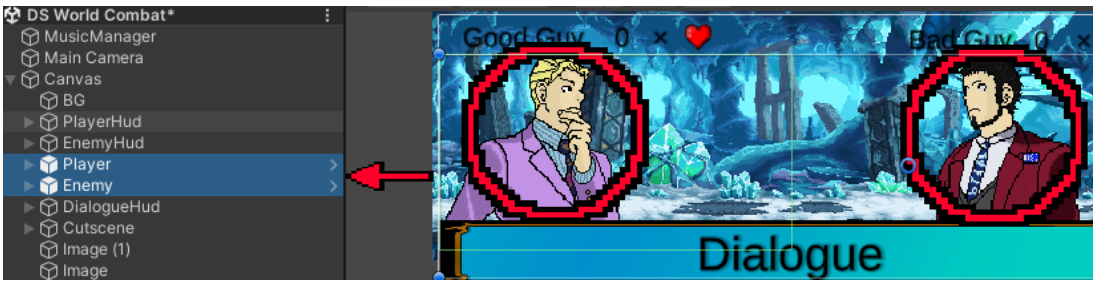

Figure 67: Image showing the `GameObjects` with the `CombatUnit` script attached. These are present in every combat scene, and represents the player and enemy characters.

Function description

- `TakeDamage`: Subtract `currentHP` variable by the input amount. Returns true if `currentHP` is at or below 0 from the damage, otherwise return false.

6.2.2.9. `CoWorkerCutscene`

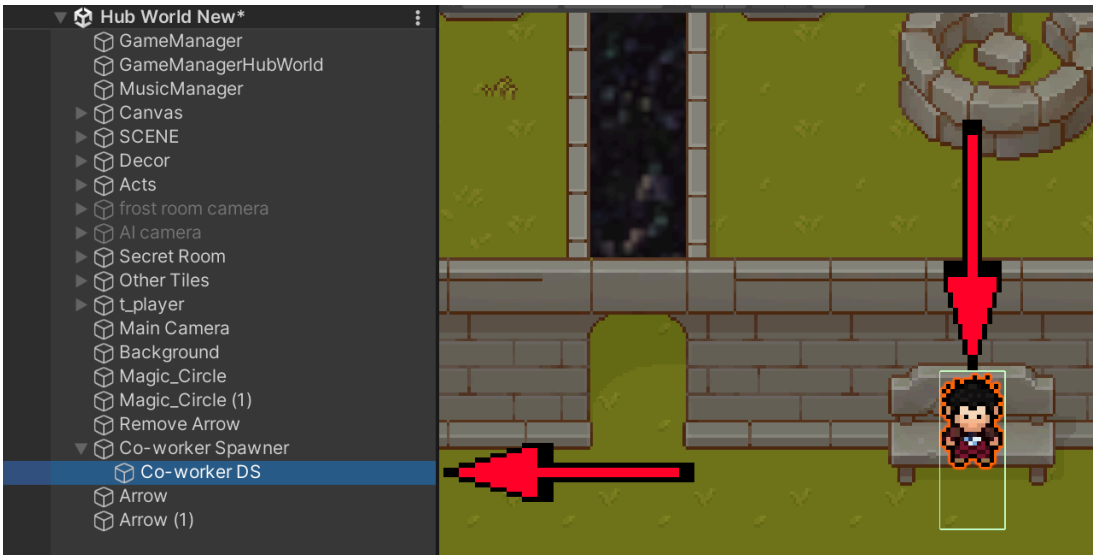

Figure 68: Image showing the `GameObjects` with the `CoWorkerSpawner` script attached.

Function description

- `Update`: Load the next dialogue box when the player presses the interact key.
- `OnTriggerEnter2D`: Check if the player collides with the `GameObject` that has this script attached and starts the dialogue if so.

6.2.2.10. `CoWorkerSpawner`

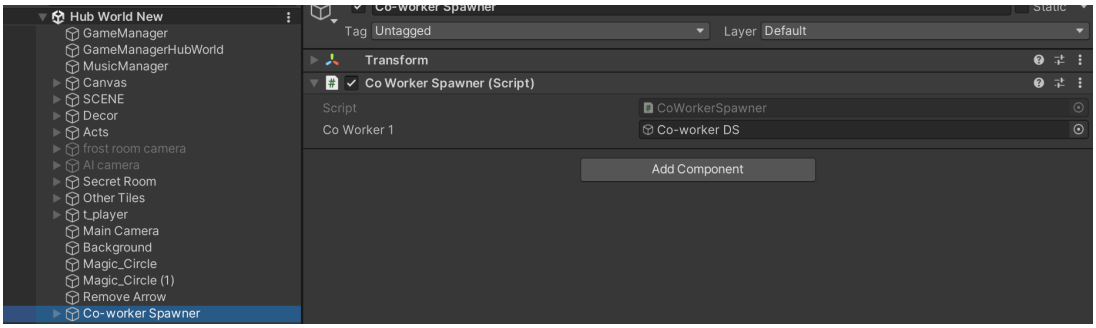

Figure 69: Image showing the `GameObjects` with the `CoWorkerSpawner` script attached.

Function description

- `Update`: Check if the item has been acquired in one of the worlds and spawn the corresponding co-worker if so.

6.2.2.11. `Credits`

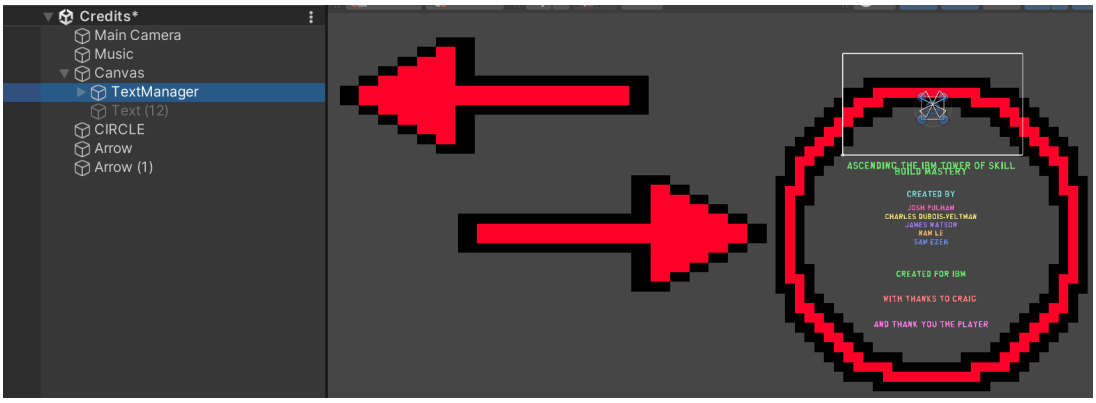

Figure 70: Image showing the `GameObject` with the `Credits` script attached.

Function description

- `Start`: Runs the `TheEnd` function.
- `Update`: Moves the credit text upwards by a small amount every frame.
- `TheEnd`: Wait for 12 seconds, then display the "The End" text. Wait for another 10 seconds after that and quits the game.

6.2.2.12. `EnableArrow`

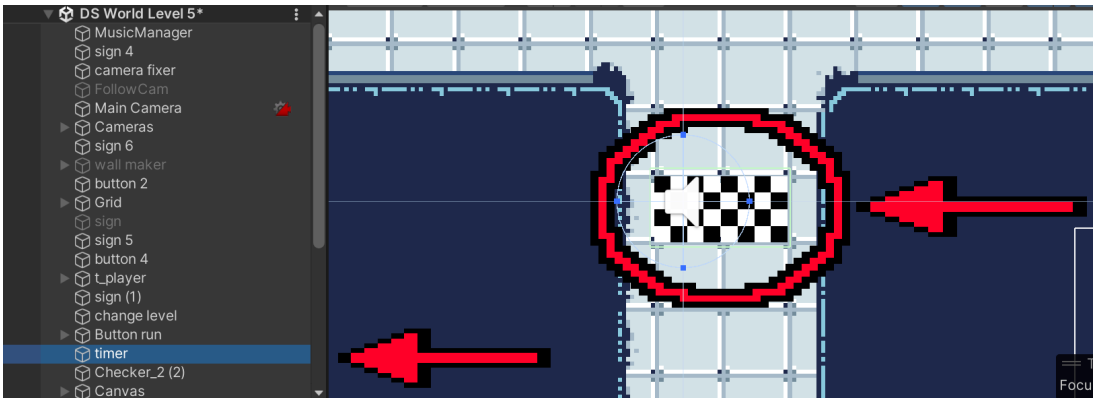

Figure 71: Image showing the `Timer` `GameObject` with the `EnableArrow` script attached in DS World Level 5. `EnableArrow` is also linked to the Altar Cutscene 3 `GameObject` in the Hub World.

Function description

- `OnTriggerEnter2D`: If the player collides with the `GameObject` holding this script, the arrow `GameObject` is enabled.

6.2.2.13. `EnemyTurn`

See Section 6.3.3 for more detail.

Function description

- `Start`: Generate a question from `QuestionSetup`'s `Generate` function, then transition to `PlayerTurn`.

6.2.2.14. `GM`

The game manager enters the game environment through the `MainMenu` and then duplicates itself across scenes.

Function description

- `Update`: The Update method is executed at the beginning of each frame in the Unity game, we use this to poll the game engine state to check whether key variables are loaded such as the audio mixer and relevant sliders and user input fields for fields such as the volume.
- `Awake`: The Awake method is called at the creation of a new GM object. As we are using the singleton pattern, we use this to check whether there is already an instance of a game manager that is globally loaded. In the case that no game manager object is loaded globally then we assign the current game manager object as the designated global instance, otherwise, we leave the global game manager instance untouched.
- `GetParticle`: Returns the user's particle preference settings.
- `SetParticle`: Sets the user's particle preference to the boolean value provided.
- `GetVolume`: Returns the game's current volume level.
- `SetVolume`: Accepts a float as input and then uses this to set the volume level associated with the audio mixer and stores this volume level to disk so that it can be recovered during the next load of the game.
- `GetGraphics`: Returns the user's graphics quality settings.
- `SetGraphics`: Accepts an integer and then sets the graphics quality level to be equal to this integer.

6.2.2.15. `ImportQuestions`

See section

Function description

- `Import`: Reads from a CSV, and creates `questionData ScriptableObject` from each CSV line in the specified Resources folder.
- `SplitCSVLine`: Parse through a line and splits on semicolons outside of quotation marks.

6.2.2.16. `Inventory`

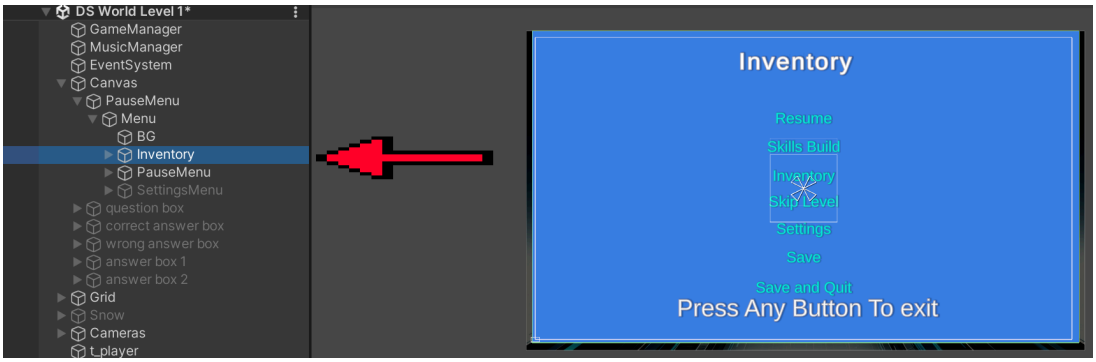

Figure 72: Image of the Inventory with the player not having acquired any items yet.

Function description

- `Update`: Checks the `Game Manager` for which items have been acquired by the player and displays the acquired items in the inventory menu. Also, checks if any input is pressed to close the inventory menu.

6.2.2.17. `Lost`

See Section 6.3.3 for more detail.

Function description

- `Start`: Display text after the player loses, calls `ButtonManager`'s `ClearButtons` function and quits the combat scene.

6.2.2.18. `MainMenuBackground`

Function description

- `Start`: Get the animation component to move the image around the screen.
- `Update`: Get a random number between 0 and 9. If a 0 is rolled, the image is blurred. If a 1, 2, 3 or 4 is rolled, the image moves around the scene whereas if a 5, 6 or 7 is rolled, the image is blurred and moves around the scene. Otherwise, nothing happens. This random number is rolled once every 4 seconds.

6.2.2.19. `MainMenuButtons`

Figure 73: The main menu of the game

Function description

- `Start`: The `Start` method is called on the first frame of the scene, we use this method to enable particle effects
- `Quit`: The `Quit` method handles the click event from the Quit button in the Main Menu. When the Quit button is clicked, the Quit method uses the `Application.Quit` function from the Unity API to quit the game.
- `Volume`: The Volume method handles input change events from the Volume slider in the Main Menu. The `Volume` method in the `MainMenuButtons` class uses the singleton Game Manager instance to globally set the volume across the game.
- `Particles`: The `Particles` method is called in response to the `Particles` settings changing in the Settings sub-menu of the Main Menu. When called, the `Particles` method forwards its argument to the singleton `GameManager` object, which then globally sets particle settings. This system allows users to specify whether they would like particles to be displayed in the game so we can ensure that users with low-end hardware can turn off computationally expensive particle rendering, which ensures that they can still have a smooth experience while playing the game.

- `FullScreen`: The `FullScreen` method toggles whether the game is to be played in full screen.
- `MainMenu`: The `MainMenu` method initialises the `MainMenu` scene by setting the components of the Main Menu to be visible to the user.
- `OptionsMenu`: The `OptionsMenu` method displays the game options that the player can control alongside sliders, toggle buttons and dropdowns to manipulate their values.
- `SetQuality`: The `SetQuality` method is called in response to changes in the value specified by the user in the Quality dropdown. When the user selects a new Graphics Quality setting, the `SetQuality` method forwards this value to the singleton Game Manager object's `SetGraphics` method to globally set the graphics quality across the game.
- `Credits`: The `Credits` method is called in response to the user clicking the Credits button. When the user clicks the Credits button, the `Credits` method uses the Unity `SceneManager` API to load the `Credits` scene by using the `LoadScene` method.

6.2.2.20. `MusicManager`

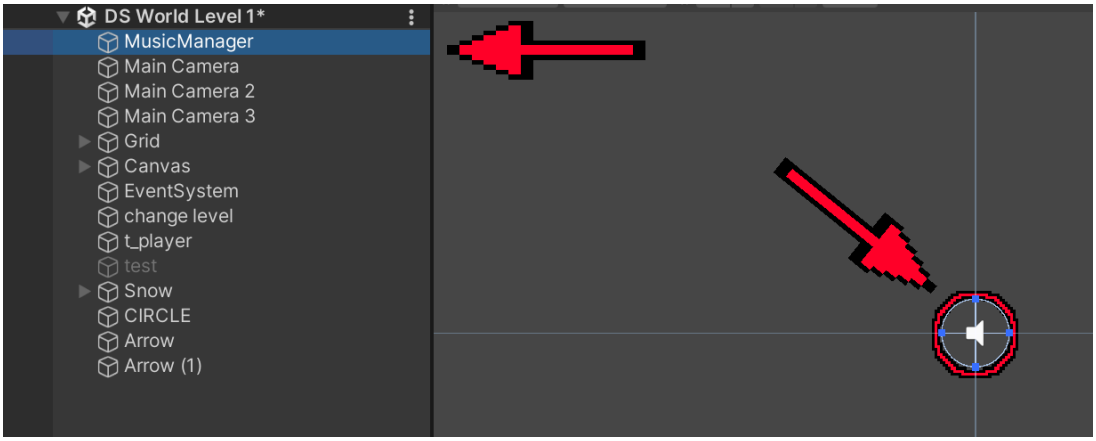

Figure 74: Image of a non-physical `GameObject` with the `MusicManager` script attached. The `MusicManager` object exists in every scene.

Function description

- `Awake`: Check if an instance of the `MusicManger` script exists in the current scene. If it does, do nothing. If it doesn't, take the instance from the last scene and import it into this scene.
- `PlayMusic`: Plays an audio clip when the function is called.
- `StopMusic`: Stops the current audio clip from running when the function is called.
- `ChangeMusic`: Updates the current playing audio clip to a different one.

6.2.2.21. `PauseMenu`

Function description

- `Start`: The `Start` method is called on the first frame of the scene, we use this method to enable particle effects and load the user's graphics quality preferences into the dropdown in the UI.
- `Update`: The `Update` method is called during every frame. We use the update method as a hook to initialise the game's settings and load them into the interface from the GameManager.
- `Resume`: The `Resume` method in the PauseMenu class responds to the user pressing the Resume button to allow them to re-enter the game after pausing it.
- `Pause`: The `Pause` method loads the Pause Menu interface onto the screen and allows the user to take a break from the game.
- `QuitGame`: The `QuitGame` method responds to the Quit button in the Pause Menu's user interface. When the Quit button is called, the PauseMenu instance uses the Unity API to exit the game.

- `MainMenu`: The `MainMenu` method is called in response to the user pressing the Main Menu button in the Pause Menu's interface. When called, the `MainMenu` method loads the Main Menu and exits the Pause Menu interface
- `OptionsMenu`: The `OptionsMenu` method loads the options into the Pause Menu interface.
- `Volume`: The `Volume` method is called in response to changes to the Volume slider in the user interface for the Pause Menu. When called, the `Volume` method propagates changes towards the singleton Game Manager instance so that Volume settings persist globally throughout the game.
- `Particles`: The `Particles` method is called in response to changes in the Particles toggle button in the Pause Menu's user interface. When the user toggles the state of the Particle setting, Unity calls the `Particles` method with the updated value which we then propagate to the singleton Game Manager instance to ensure that particle settings persist throughout the game.
- `SkillsBuild`: The `SkillsBuild` method uses the Unity API to open up the IBM Skills Build website in the player's browser
- `SkipLevel`: The `SkipLevel` method instructs the to skip the current level that the player is in
- `DisplayText`: The `DisplayText` method displays the combat text from the current scene
- `DisableText`: The `DisableText` method is a coroutine that hides the combat text after a delay.
- `WaitForSecondsRealtime`: The `WaitForSecondsRealTime` method is a coroutine that removes the save text in the Pause Menu's User Interface after a delay.
- `GetSceneNameByBuildIndex`: `GetSceneNameByBuildIndex` is static method that takes the build index of a scene as an argument and returns the corresponding scene name.
- `GetSceneNameFromScenePath`: The static `GetSceneNameFromScenePath` method takes the path of a scene as an argument and mechanically produces the name of the scene.
- `SetQuality`: The `SetQuality` method is bound the the Graphiccs Quality drop down in the User Interface for the Pause Menu. When the player selects a new graphics quality level in the User interface.
- `Inventory`: The `Invetory` method displays the user's inventory
- `Save`: The `Save` method saves the game state to disk to later be reloaded by.
- `RemoveSaveText`: The `RemoveSaveText` button removes the Save text.
- `WaitForSecondsRealtime`: The `WaitForSecondsRealtime` method is a corountine that inserts a delay of 3 seconds.

6.2.2.22. `PlayerTurn`

See Section 6.3.3 for more detail.

Function description

- `Answer`: Depending on input, run `CombatUnit`'s `TakeDamage` function on either the player unit or the enemy unit. Both actions update the HUD using the `SetHp` function for HP, changes `BattleSystem`'s `dialogue` variable for text display, and changes the player and enemy sprites with `GameObject` transforms. After resolving the damage, go to the `Won` or `Lost` state, depending on whether the enemy or the player is dead. If none of these occur, transition to `EnemyTurn`.

6.2.2.23. `PostBattleCutscene`

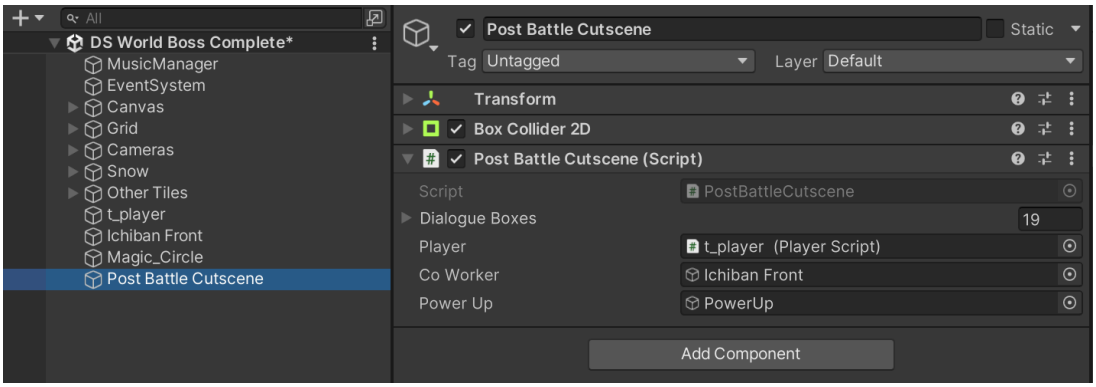

Figure 75: Image of the Post Battle Cutscene `GameObject` with the `PostBattleCutscene` script attached.

Function description

- `Update`: Load the next dialogue box when the player presses the interact key. If there are no more dialogue boxes to load, then the `GameObject` holding this script is destroyed.
- `OnTriggerEnter2D`: Check if the player collides with the `GameObject` that has this script attached and starts the dialogue if so.

6.2.2.24. `PowerUp`

Figure 76: Image of the power-up selection screen, where the player can choose which stat to upgrade.

Function description

- `Attack`: Increases the player's attack stat by a specified amount.
- `Hp`: Increases the player's hp stat by a specified amount.

6.2.2.25. `PreBattleCutscene`

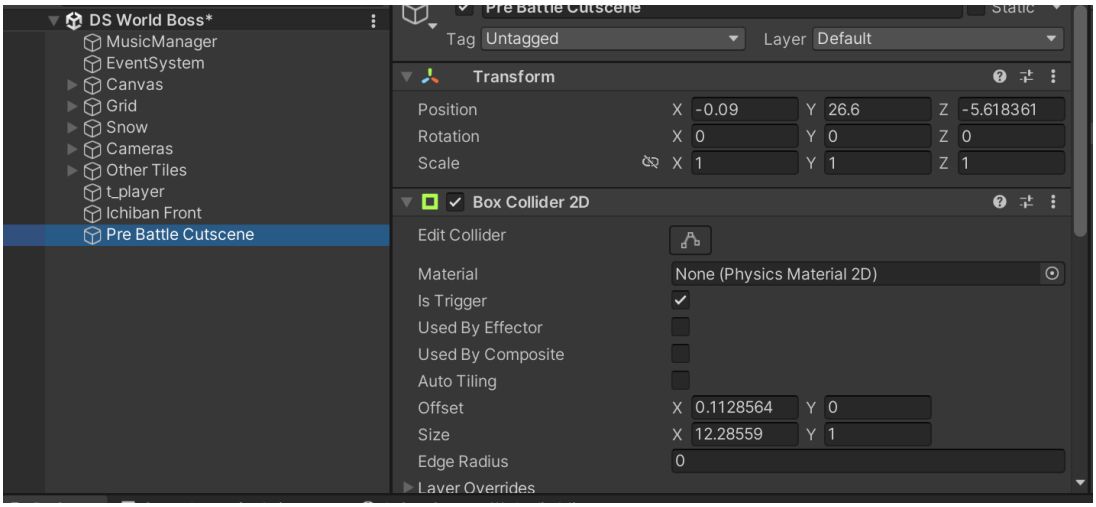

Figure 77: Image of the pre-battle cutscene `GameObject` with the `PreBattleCutscene` script attached.

Function description

- `Update`: Load the next dialogue box when the player presses the interact key. If there are no more dialogue boxes to load, then the combat level is loaded.
- `OnTriggerEnter2D`: Check if the player collides with the `GameObject` that has this script attached and starts the dialogue if so.

6.2.2.26. `QuestionData`

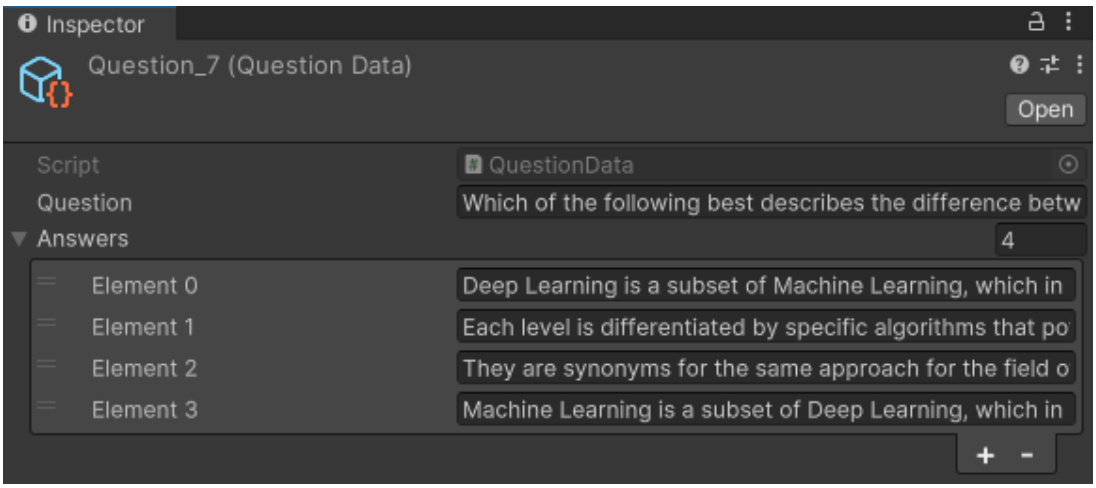

Figure 78: Image of an instantiated `QuestionData` object. The object contains a `Question` field, and an `Answers` field. The first element in the answers field is always correct.

Function description

None

6.2.2.27. `QuestionSetup`

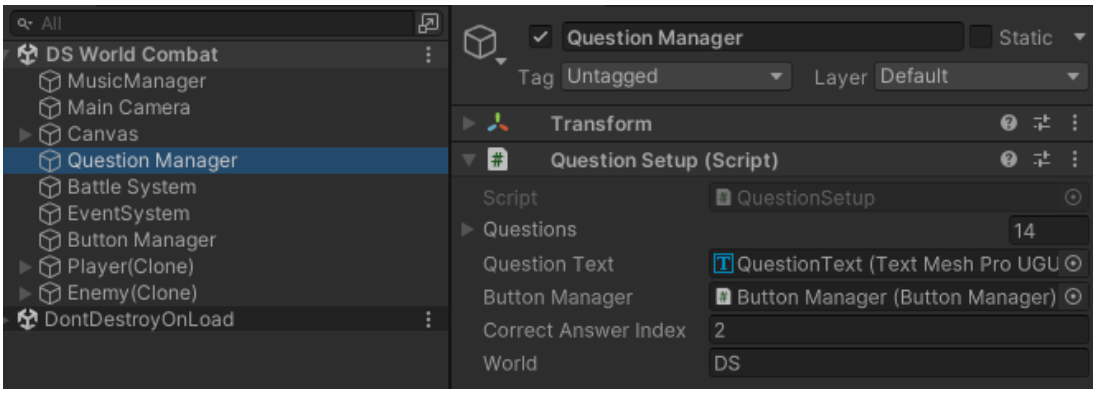

Figure 79: Image showing the `GameObject` with the `QuestionSetup` script attached. This is present in every combat scene, but is invisible. To select the questions used, simply change the `World` variable to the name of the folder which contains your questions in "Resources".

Function description

- `Awake`: Call the `GetQuestionAssets` function and create a `usedQuestions` list to be used later.
- `GetQuestionAssets`: Load all `QuestionData` from the given `world` file.
- `Generate`: Call `SelectNewQuestion`, display the buttons via the `ButtonManager`'s `DisplayButtons` function, update the question text and then call `SetAnswerValues`.
- `SelectNewQuestion`: Pop a random question off from the `questions` array to update the `currentQuestion` and `numberOfAnswers` variables with, then add it to `usedQuestions`.
- `SetAnswerValues`: Call `RandomizeAnswers`, then set the answers, and whether it is correct or not on the answer buttons through `ButtonManager`.
- `RandomizeAnswers`: Shuffle the original answers list, while keeping track of where the correct answer is in the `correctAnswerIndex` variable. Return a shuffled list of answers.

6.2.2.28. `Sign`

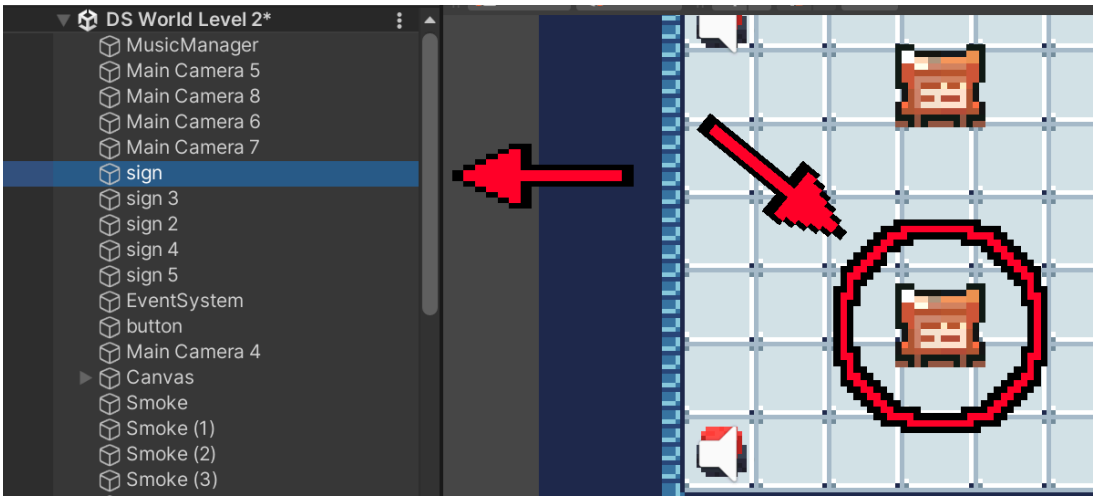

Figure 80: Image showing the `GameObjects` with the `Sign` script attached. These objects can be interacted with by the player.

Function description

- `Update`: If the player is in range of `GameObject` and presses the interact key, a dialogue box is enabled.

- `OnTriggerEnter2D`: Check if the player collides with `GameObject` and if this is the case, the player is in range.
- `OnTriggerExit2D`: Check if the player leaves the collision range and if this is the case, the player is no longer in range and the dialogue box is disabled.

6.2.2.29. `State`

See Section 6.3.3 for more detail.

Function description

- `Start`: Empty function to override. Used whenever a state is instantiated.
- `Answer`: Empty function to override. Used for answering questions in combat.

6.2.2.30. `StateMachine`

See Section 6.3.3 for more detail.

Function description

- `SetState`: Set the current `State` variable to the input state, then run the current state's `Start` function.

6.2.2.31. `Won`

See Section 6.3.3 for more detail.

Function description

- `Start`: Display text after the player wins, call `ButtonManager`'s `ClearButtons` function and quits the combat scene.

6.2.2.32. `AltarCutscene1`

Figure 81: Image showing the `GameObjects` with the `AltarCutscene1` script attached. When entering one of the locations in the image above in the Hub World, during Act 1, a cutscene will be triggered.

Function description

- `Update`: Load the next dialogue box when the player presses the interact key. If there are no more dialogue boxes to load, destroy the `GameObject`.
- `OnTriggerEnter2D`: Check if the player collides with `GameObject` and if this is the case, load the first dialogue box.

6.2.2.33. `AltarCutscene2`

Figure 82: Image showing the `GameObject` with the `AltarCutscene2` script attached. When moving onto the altar during Act 1, a cutscene will be triggered.

Function description

- `Start`: Get the `BoxCollider2D` component from `GameObject` using this script.
- `Update`: Load the next dialogue box when the player presses the interact key.
- `OnTriggerEnter2D`: Check if the player collides with `GameObject` and if this is the case, load the first dialogue box.

6.2.2.34. `AltarCutscene3`

Figure 83: Image showing the `GameObjects` with the `AltarCutscene3` script attached. When entering the location in the image above in the Hub World, during Act 1, a cutscene will be triggered.

The `Altar Cutscene 3` `GameObject` is not a physical object and is enabled during Act 1.

Function description

- `Start`: Get the `BoxCollider2D` component from `GameObject` using this script.
- `Update`: Load the next dialogue box when the player presses the interact key. If there are no more dialogue boxes to load, destroy the `GameObject`. When the 6th dialogue box is loaded, enable the arrow `GameObject`.
- `OnTriggerEnter2D`: Check if the player collides with `GameObject` and if this is the case, load the first dialogue box.

6.2.2.35. `AltarCutscene4`

Figure 84: Image showing the `GameObject` with the `AltarCutscene4` script attached. When entering the location in the image above in the Hub World, during Act 2, a cutscene will be triggered. This script is also used by `GameObject` `Altar Cutscene 5/6/7`.

Function description

- `Start`: Get the `BoxCollider2D` component from `GameObject` using this script.
- `Update`: Load the next dialogue box when the player presses the interact key. When the first dialogue box is loaded, start the music track.
- `OnTriggerEnter2D`: Check if the player collides with `GameObject` and if this is the case, load the first dialogue box.

6.2.2.36. `AltarItemGet`

Figure 85: Image showing the `GameObjects` with the `AltarItemGet` script attached. When entering one of the locations in the image above in the Hub World, a cutscene will be triggered similar to `AltarCutscene1`.

Function description

- `Update`: Load the next dialogue box when the player presses the interact key. If there are no more dialogue boxes to load, destroy the `GameObject`.
- `OnTriggerEnter2D`: Check if the player collides with `GameObject` and if this is the case, load the first dialogue box.

6.2.2.37. `CreepyCutscene`

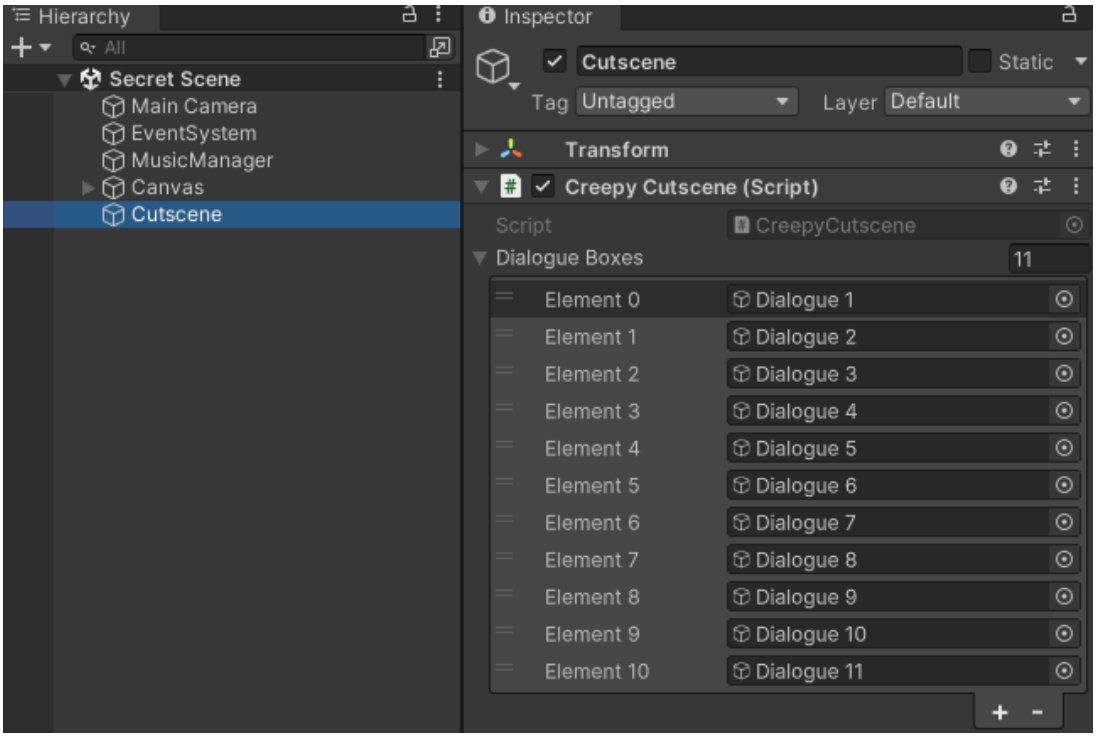

Figure 86: Image showing the `GameObject` with the `CreepyCutscene` script attached. This script is run on scene entrance.

Function description

- `Update`: Load the next dialogue box when the player presses the interact key. When the last dialogue box has been reached, force quit the game and load the URL link.

6.2.2.38. `HubDoor`

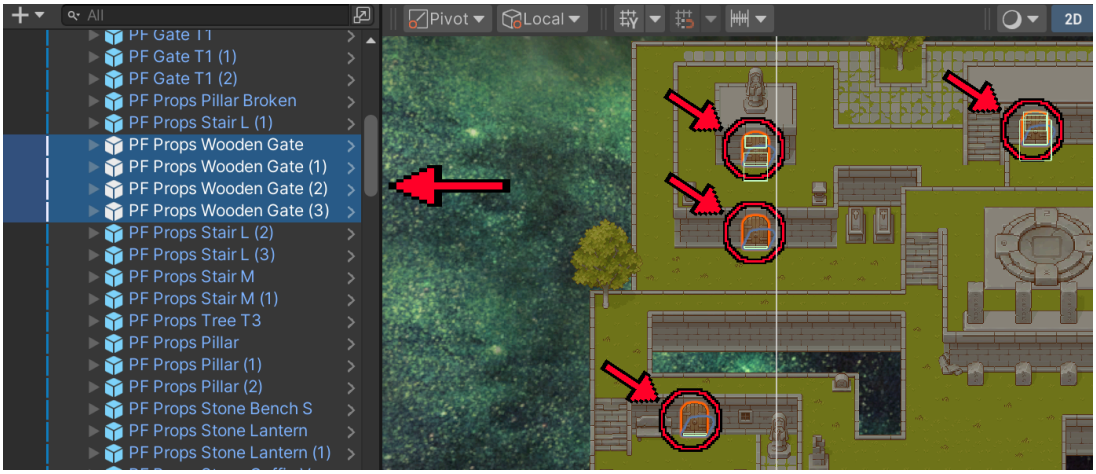

Figure 87: Image showing the `GameObjects` with the `HubDoor` script attached. 2 of the 4 doors, currently do not have a connected room, as we have only created 2 worlds so far.

Function description

- `Update`: Check if the player is in range of the `GameObject` holding this script and if so and the interact key is pressed the door sprite changes to that of an open door and the door no longer has collision so that the player can enter.

- `OnTriggerEnter2D`: Check if the player collides with `GameObject` and if this is the case, the player can press the interact key.
- `OnTriggerExit2D`: Check if the player leaves the collision range of the `GameObject` and if this is the case, the player can no longer press the interact button, the door sprite changes to a closed door and the door's collision is enabled.

6.2.2.39. `HubDoorTrigger`

Figure 88: Image showing the `GameObjects` with the `HubDoorTrigger` script attached. These are child objects of the `Wood Gate GameObject` that holds the `HubDoor` script.

Function description

- `OnTriggerEnter2D`: Check if the player collides with the `GameObject` holding this script and if this is the case, the player is moved to a specified location (a different room), the current camera is disabled and the next room's camera is enabled.

6.2.2.40. `HubTeleport`

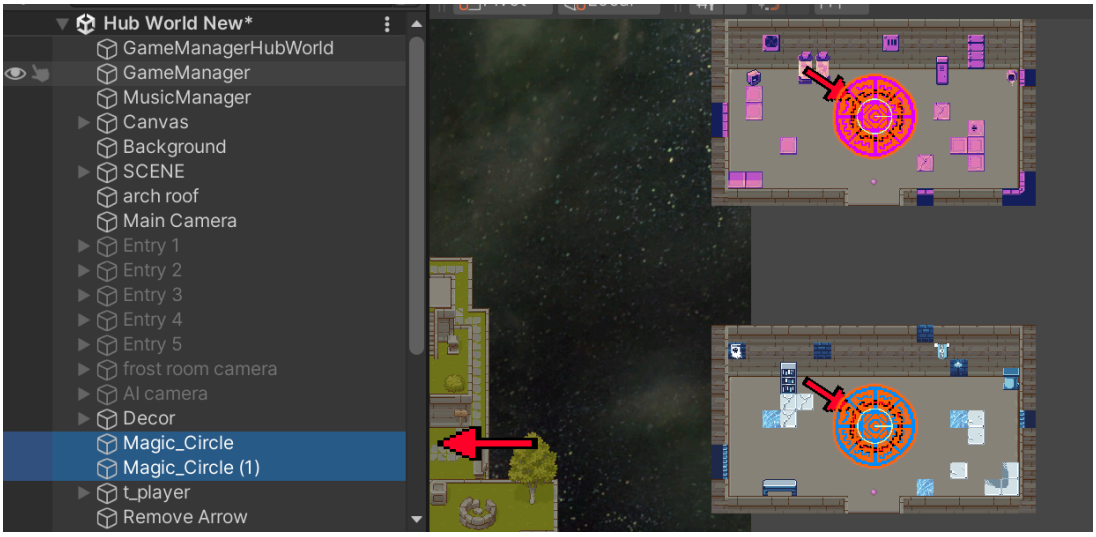

Figure 89: Image showing the `GameObjects` with the `HubTeleport` script attached. Entering the `Magic_Circle` will load another scene.

Function description

- `OnTriggerEnter2D`: Check if the player collides with the `GameObject` holding this script and if this is the case, a specified scene is loaded.

6.2.2.41. `IntroCutscene`

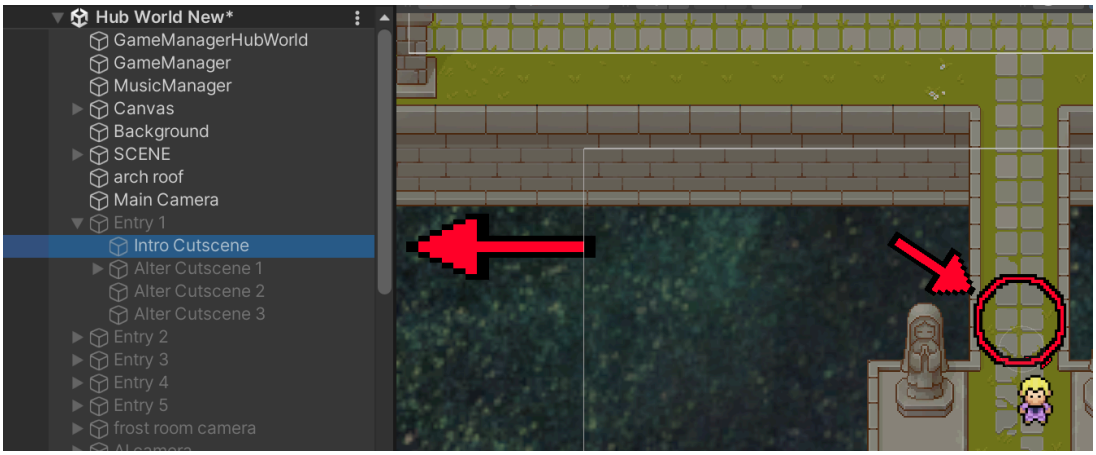

Figure 90: Image showing the `GameObjects` with the `IntroCutscene` script attached. When entering the location in the image shown above, the cutscene will be triggered.

Function description

- `Update`: Load the next dialogue box when the player presses the interact key. If there are no more dialogue boxes to load, destroy the `GameObject`.
- `OnTriggerEnter2D`: Check if the player collides with `GameObject` and if this is the case, load the first dialogue box.

6.2.2.42. `GMHub`

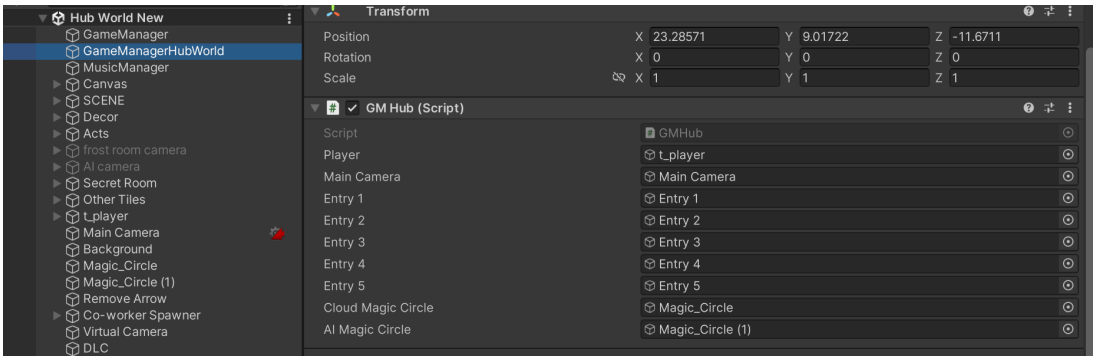

Figure 91: Image showing the `GameMangerHubWorld` `GameObjects` with the `GMHub` script attached.

Function description

- `Start`: Check the number of items acquired and change the `Act` depending on how many have been acquired. Also disables the portal from worlds already entered

6.2.2.43. `HubBackground`

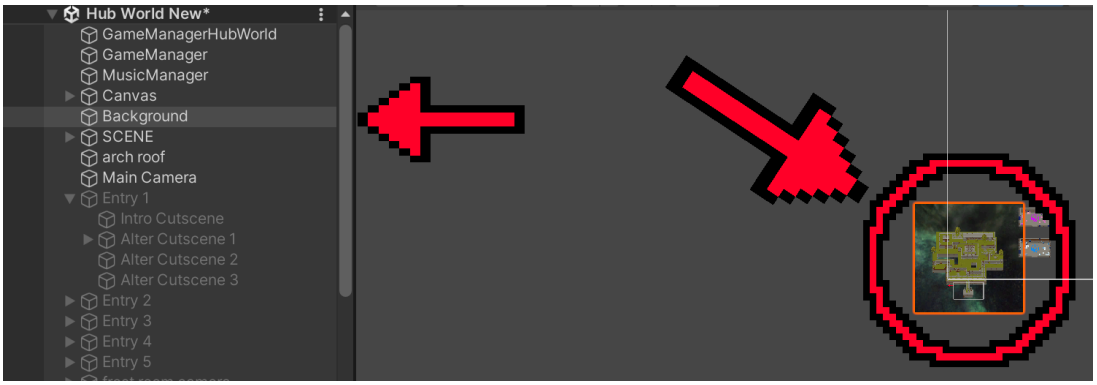

Figure 92: Image showing the `GameObjects` with the `HubBackground` script attached. This is simply an image object.

Function description

- `Update`: Rotate the background object on the z-axis by a set speed every frame.

6.2.2.44. `NamePlayer`

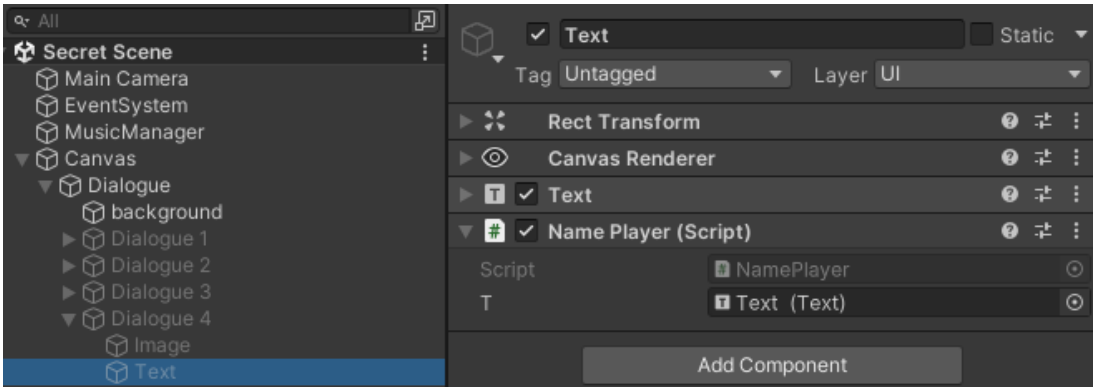

Figure 93: Image showing the `NamePlayer` script attached to a Text `GameObject`. This is not a physical `GameObject`

Function description

- `Start`: Access the player's real-life name by checking their system environment's name (files).

6.2.2.45. `Portal`

Figure 94: Image of `Portal` script attached to Portal `GameObject` in Hub World. This object is enabled during act 5.

Function description

- `Start`: Gets the `BoxCollider2D` component from the `GameObject` holding this script. Gets the `SpriteRenderer` component from the `GameObject` holding this script. Disables the `BoxCollider2D` component and disables the `SpriteRenderer` component
- `Update`: If the specified cutscene is done, enable both the `BoxCollider2d` and `SpriteRenderer` components.

6.2.2.46. `Button`

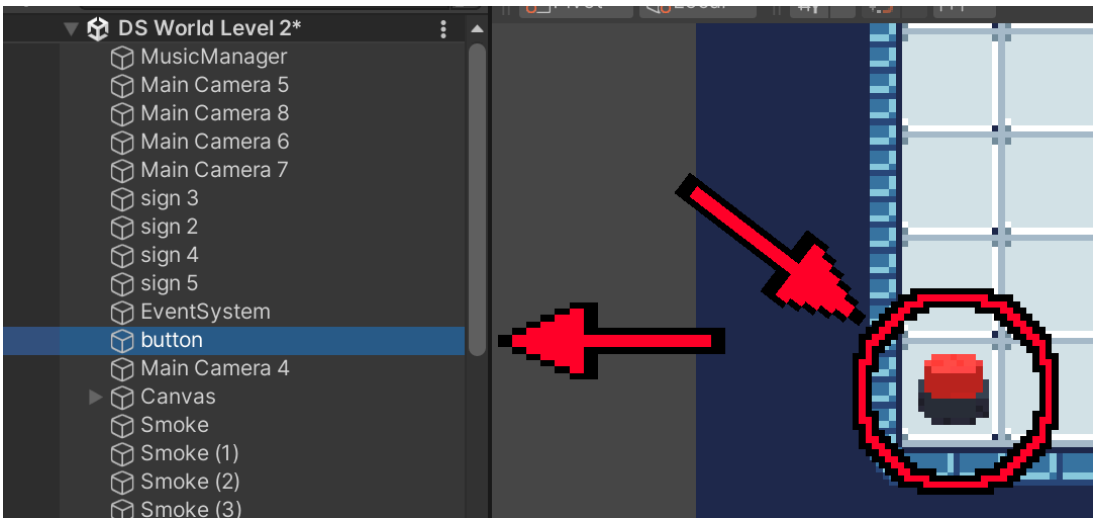

Figure 95: Image of Button script attached to a Button `GameObject` in DS World Level 2. The `button` `GameObject` exists in every level where the player must answer a question to continue.

Function description

- `Start`: Get `AudioSource` Component.
- `Update`: If the player is in range of the button `GameObject`, presses the interact key and the button corresponds to a correct answer, then the `OpenDoor` function is run, the wall `GameObject` is disabled and dialogue is shown. If the button corresponds to an incorrect answer, then the `Return` function is run and dialogue is shown.
- `OnTriggerEnter2D`: Check if the player collides with `GameObject` and if this is the case, the player can press the interact key.
- `OnTriggerExit2D`: Check if the player leaves the collision range of the `GameObject` and if this is the case, the player can no longer press the interact button.
- `Return`: Waits for 2 seconds, then destroys the `GameObject` that is attached to this script and disables the corresponding dialogue.
- `OpenDoor`: Waits for 2 seconds, then destroys the `GameObject` attached to this script and disables the corresponding dialogue.

6.2.2.47. `ButtonSpawner`

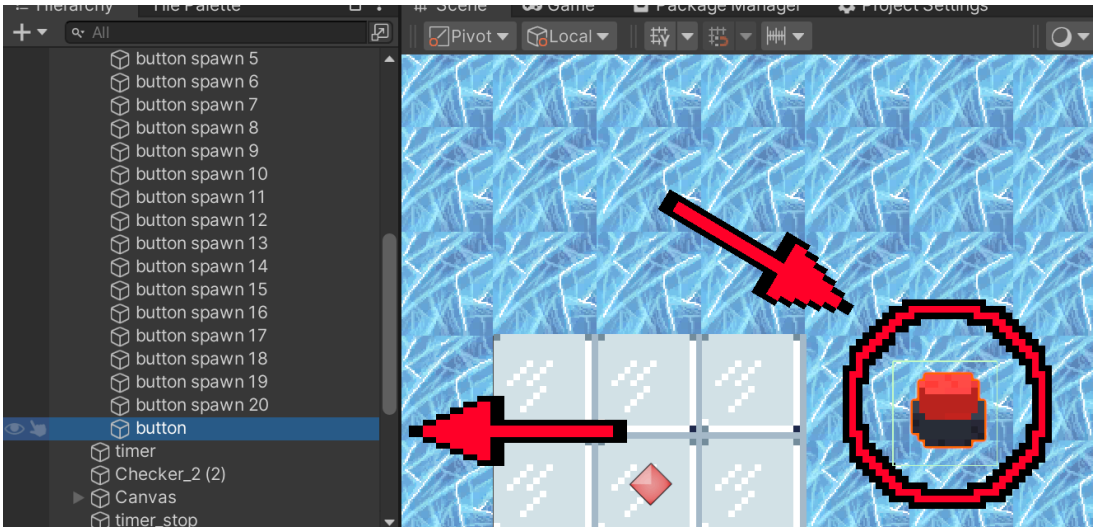

Figure 96: Image of `ButtonSpawner` script attached to Button `GameObject` in DS World Level 5.

Function description

- `Start`: Generates a random number between 0 and 19, then runs the Spawn Function.
- `Update`: Check if the player is in range of the `GameObject` holding this script and if so and the interact key is pressed the specified number of times the button was hit updates and then the `Spawn` function is run with a new random number. If the specified number of hits is reached, then the player `GameObject` is teleported to a specified location.
- `Spawn`: Sets the button `GameObjects` spawn location to that of the index in the location list of the random number.
- `OnTriggerEnter2D`: Check if the player collides with `GameObject` and if this is the case, the player can press the interact key.
- `OnTriggerExit2D`: Check if the player leaves the collision range of the `GameObject` and if this is the case, the player can no longer press the interact button.

6.2.2.48. `CameraFollow`

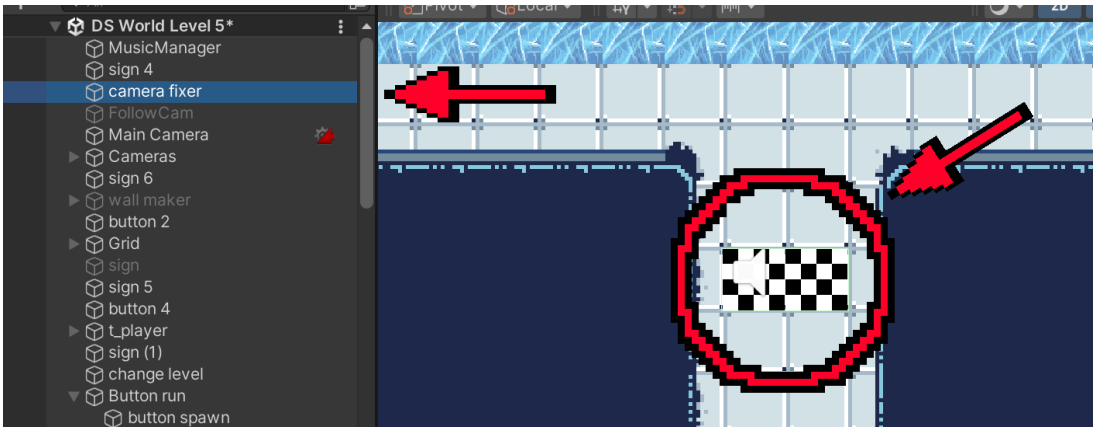

Figure 97: Image of `CameraFollow` script attached to camera fixer `GameObject` in DS World Level 5. This is not a physical `GameObject`, but contains a collision trigger that triggers the script to run.

Function description

- `OnTriggerEnter2D`: If the player collides with the `GameObject` holding this script, the follow camera is enabled and the stationary cameras are disabled.

6.2.2.49. `CameraManager`

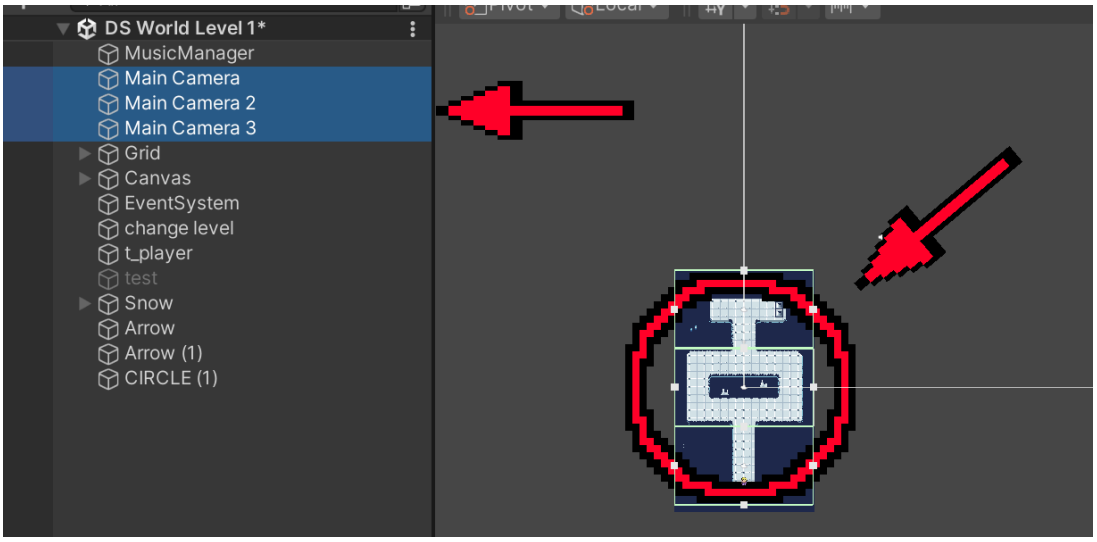

Figure 98: Image of `CameraManager` script attached to multiple Main Camera `GameObject` in DS World Level 1. Every Level that contains cameras that do not follow the player have this script attached to it. This exists in the DS and AI world.

Function description

- `OnTriggerEnter2D`: If the player collides with the `GameObject` holding this script, the camera `GameObject` is enabled.
- `OnTriggerExit2D`: If the player leaves the collision range of the `GameObject` holding this script the camera `GameObject` is disabled.

6.2.2.50. `DoorTriggerLeft`

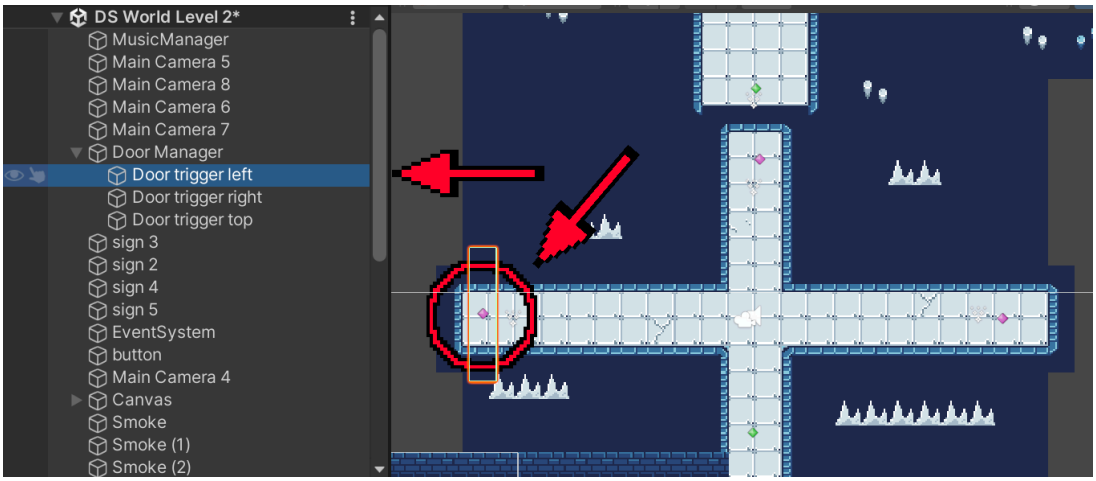

Figure 99: Image of `DoorTriggerLeft` script attached to a Door trigger left `GameObject` in DS World Level 2. This script also exists in DS World Level 4.

Function description

- `OnTriggerEnter2D`: If the player collides with the `GameObject`, set the `was_hit` string of the `LabyrinthDoor` script to "left", indicating that the left door has been interacted with.

6.2.2.51. `DoorTriggerRight`

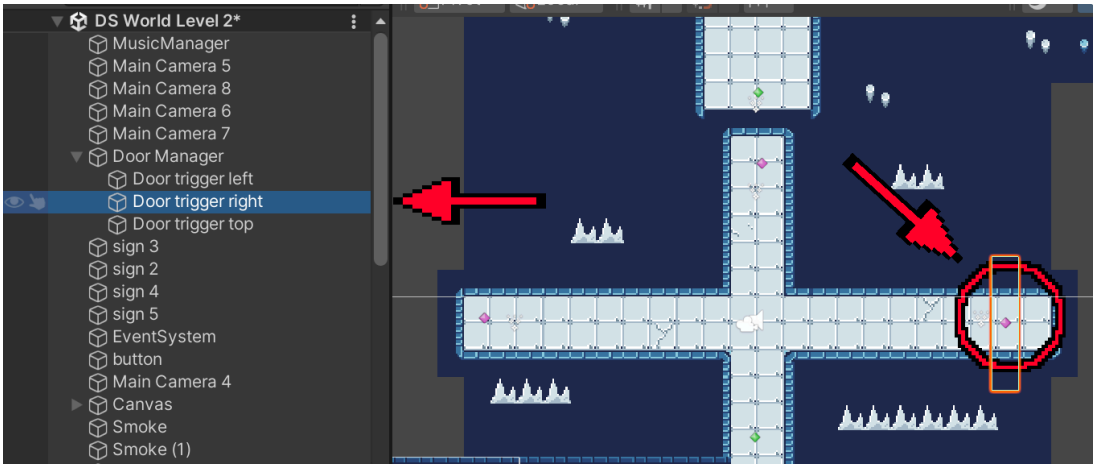

Figure 100: Image of `DoorTriggerRight` script attached to a Door trigger right `GameObject` in DS World Level 2. This script also exists in DS World Level 4.

Function description

- `OnTriggerEnter2D`: If the player collides with the `GameObject`, set the `was_hit` string of the `LabyrinthDoor` script to "right", indicating that the right door has been interacted with.

6.2.2.52. `DoorTriggerTop`:

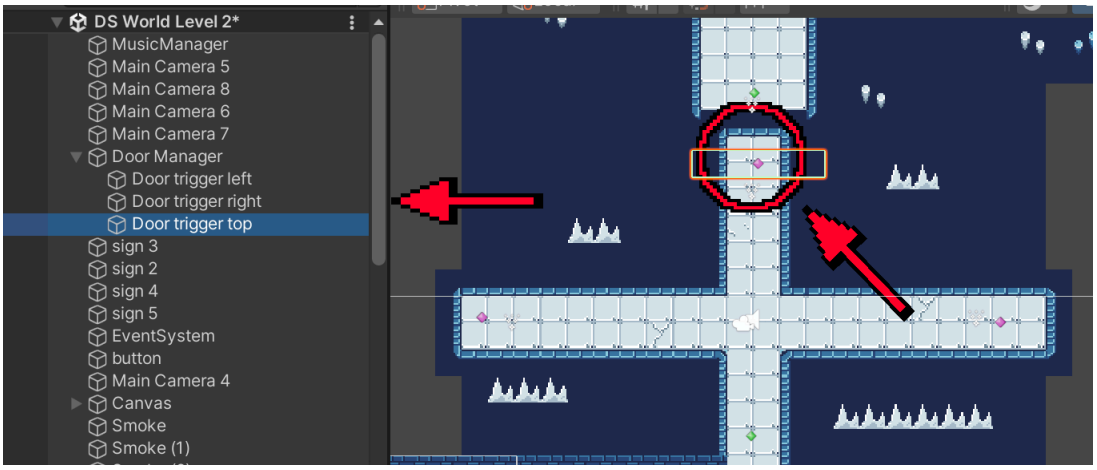

Figure 101: Image of `DoorTriggerTop` script attached to a Door trigger top `GameObject` in DS World Level 2. This script also exists in DS World Level 4.

Function description

- `OnTriggerEnter2D`: If the player collides with the `GameObject`, set the `was_hit` string of the `LabyrinthDoor` script to "top", indicating that the top door has been interacted with.

6.2.2.53. `Hole`

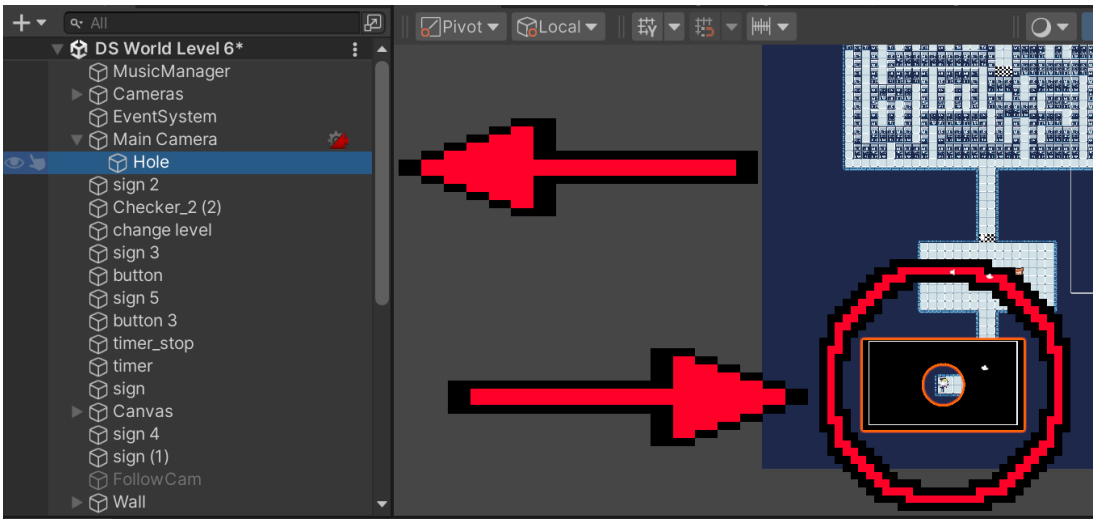

Figure 102: Image of `Hole` script attached to `Hole` `GameObject` in DS World Level 6. The `Hole` script is attached to a child object of the main camera in DS World Level 6. The child object is initially disabled.

Function description

- `OnTriggerEnter2D`: If the player collides with the `GameObject` holding this script, the whole `GameObject` is enabled.

6.2.2.54. `LabyrinthDoor`

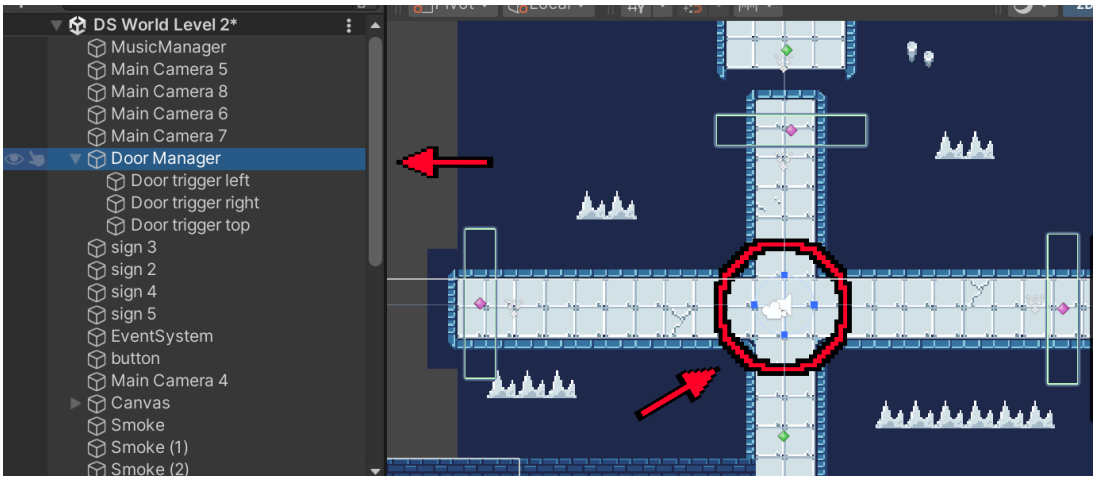

Figure 103: Image of `LabyrinthDoor` script attached to a Door Manager `GameObject` in DS World Level 2. The `Door manager` object is the parent object of the `GameObjects` containing the `DoorTriggerLeft`, `DoorTriggerRight` and `DoorTriggerTop` scripts.

Function description

- `OnTriggerEnter2D`: Check if the player collides with `GameObject` and if this is the case, get the current string out of a list of strings corresponding to either "top", "left" or "right". If this string is equal to that of the door interacted with (left door = "left", right door = "right", top door = "top"), which is got from the `DoorTrigger` scripts, then increase the counter by one. Once the counter reaches a specified maximum, set the `finished` boolean to true.

6.2.2.55. `LevelChanger`

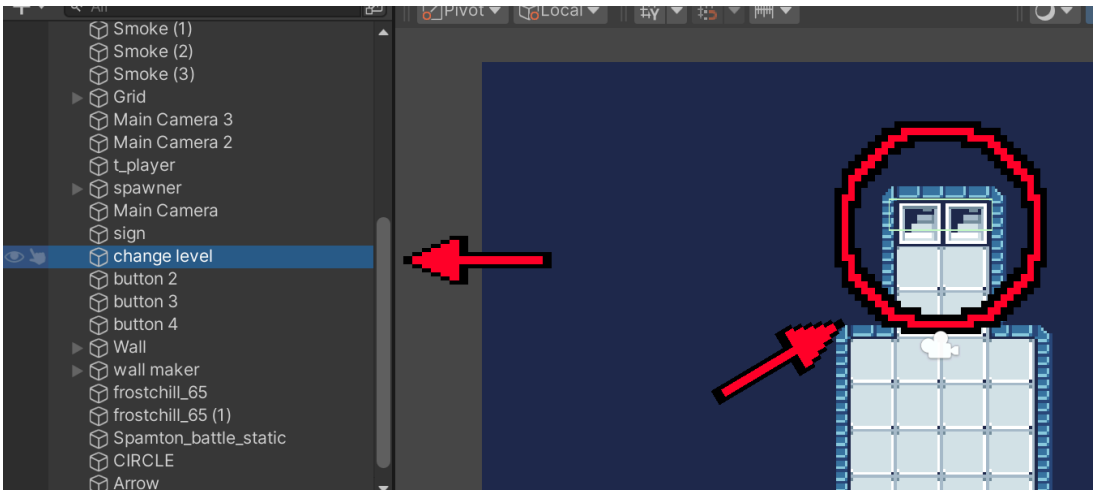

Figure 104: Image of `LevelChanger` script attached to a `change level` `GameObject` in DS World Level 2. This script exists in every DS world.

Function description

- `OnTriggerEnter2D`: Check if the player collides with `GameObject` and if this is the case, load the next scene ID. Scene ID is set in the build settings.

6.2.2.56. `PlayerIceScript`

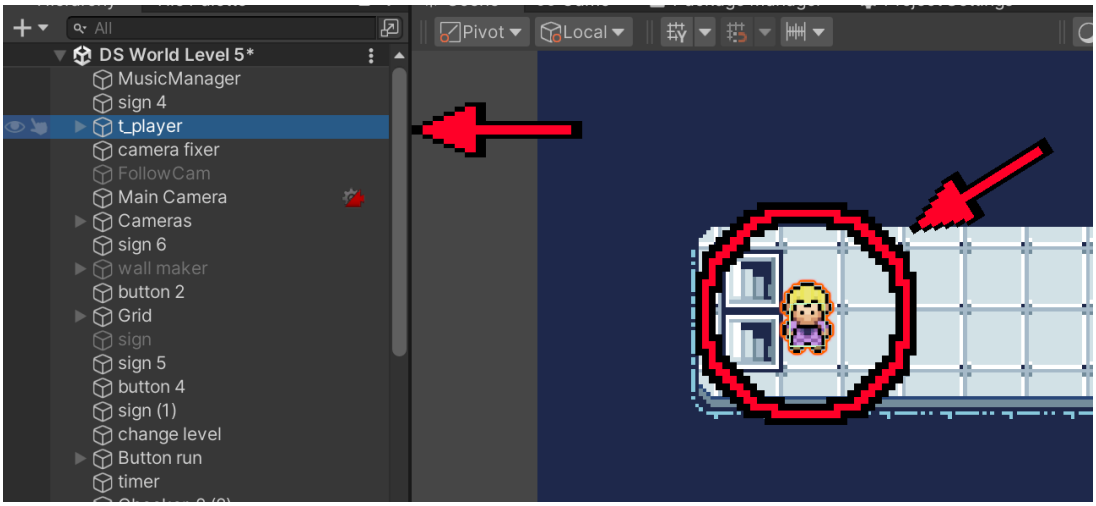

Figure 105: Image of `PlayerIceScript` script attached to the player `GameObject` in DS World Level 5. The `player` game object holds this script only in DS World Level 5.

Function description

- `Update`: Updates the player's position based on the directional input keys pressed, and updates the player's animation sprites based on this movement. It also disables the players' movement when on ice.
- `FixedUpdate`: Physically moves the player rather than just updating the movement values.
- `Flip`: Flips the player sprite when moving left.
- `OnTriggerEnter2D`: Check if the player collides with a `GameObject` of type ice and if this is the case, disables the movement input.
- `OnTriggerExit2D`: Check if the player leaves the collision range of the ice `GameObjects` and if this is the case, re-enable the player's movement input.

6.2.2.57. `PlayerScript`

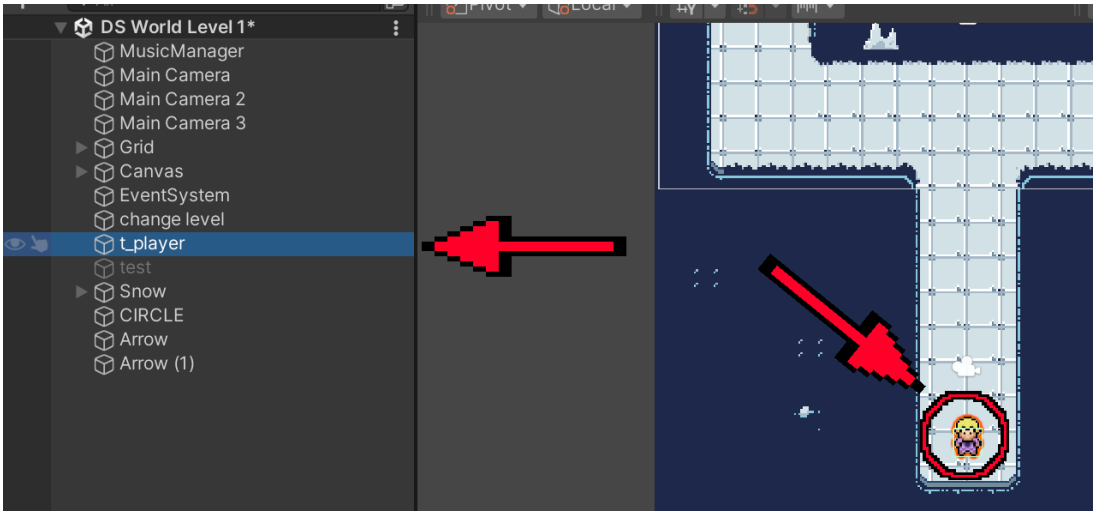

Figure 106: Image of `PlayerScript` script attached to the player `GameObject` in DS World Level 1. This script is held by the player in all levels in DS World and Hub World

Function description

- `Update`: Updates the player's position based on the directional input keys pressed, and updates the player's animation sprites based on this movement. It also disables the player's movement when on ice.
- `FixedUpdate`: Physically moves the player rather than just updating the movement values.
- `Flip`: Flips the player sprite when moving left.
- `OnTriggerEnter2D`: Check if the player collides with `GameObject` of type ice and if this is the case, disables the movement input for some time.
- `OnTriggerExit2D`: Check if the player leaves the collision range of the ice `GameObjects` and if this is the case, re-enable the player's movement input.
- `Unfreeze`: Waits for a specified amount of seconds, then re-enables the player's movement input.
- `Frozen`: Waits for 5 seconds, then reloads the current scene.

6.2.2.58. `RemoveArrow`

Figure 107: Image of `RemoveArrow` script attached to a `Remove Arrow GameObject` in Hub World. The `Remove arrow GameObject` is not a physical object, but has 2 collision triggers attached that activate the script's function.

Function description

- `OnTriggerEnter2D`: Check if the player collides with `GameObject` and if this is the case, destroy the arrow `GameObject`.

6.2.2.59. `SecretRoom`

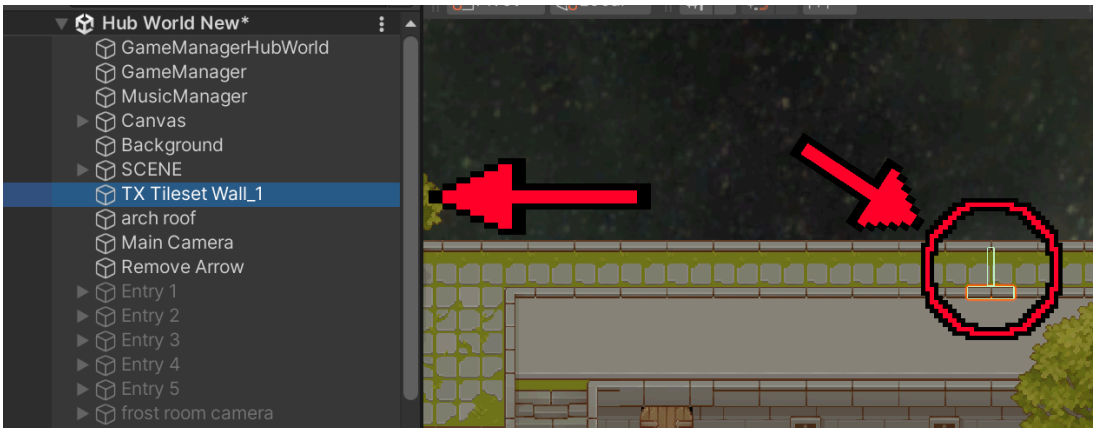

Figure 108: Image of `SecretRoom` script attached to a `TX Tileset Wall_1` `GameObject` in Hub World.

Function description

- `Update`: If the player is in range of `GameObject`, then disable its physical collision.
- `OnTriggerEnter2D`: Check if the player collides with `GameObject` and if this is the case, the player is in range.
- `OnTriggerExit2D`: Check if the player leaves the collision range and if this is the case, the player is no longer in range and the physical collision is re-enabled.

6.2.2.60. `Snow`

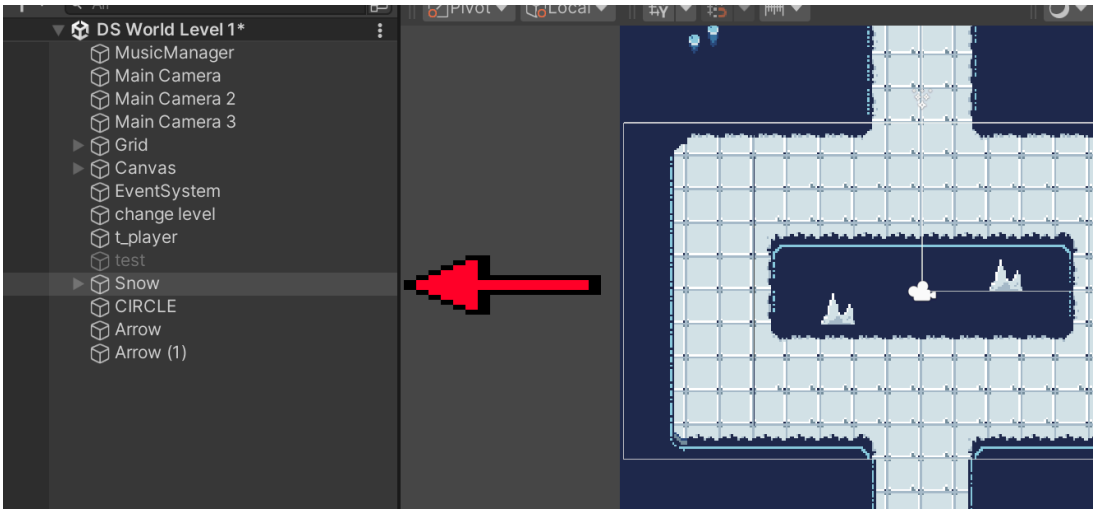

Figure 109: Image of `Snow` script attached to a `Snow` `GameObject` in DS World Level 1. `Snow` is not a physical `GameObject`.

Function description

- `Start`: Gets the child `GameObject`.
- `Update`: If particle effects are ever disabled in the Pause Menu, then the child `GameObject` is disabled.

6.2.2.61. `SnowGenerator`

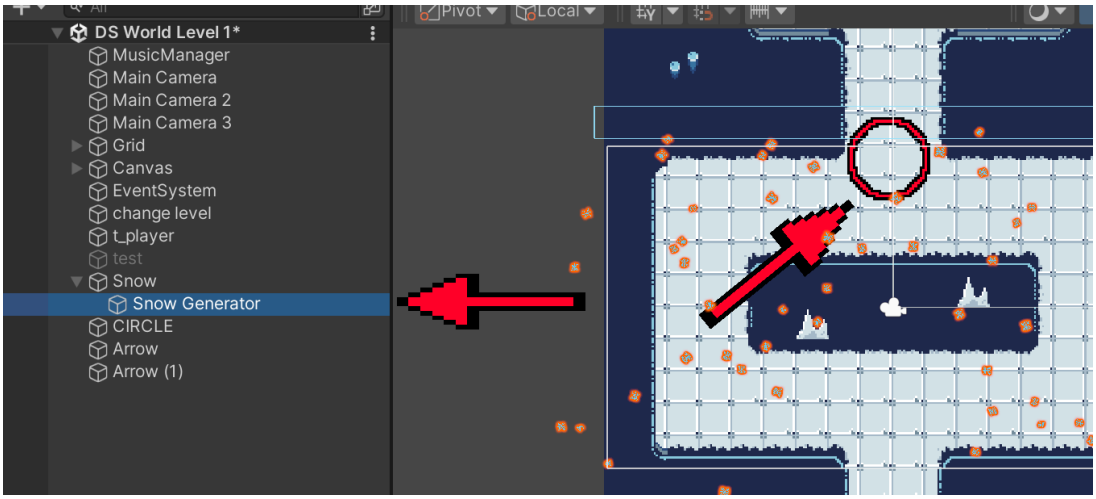

Figure 110: Image of `SnowGenerator` script attached to a `Snow Generator` `GameObject` in DS World Level 1. The `Snow generator` `GameObject` is a child object of the `Snow` `GameObject`.

Function description

- `Update`: Checks the currently active camera and move the `snow generator` `GameObject` above the currently active camera.

6.2.2.62. `Spawner`

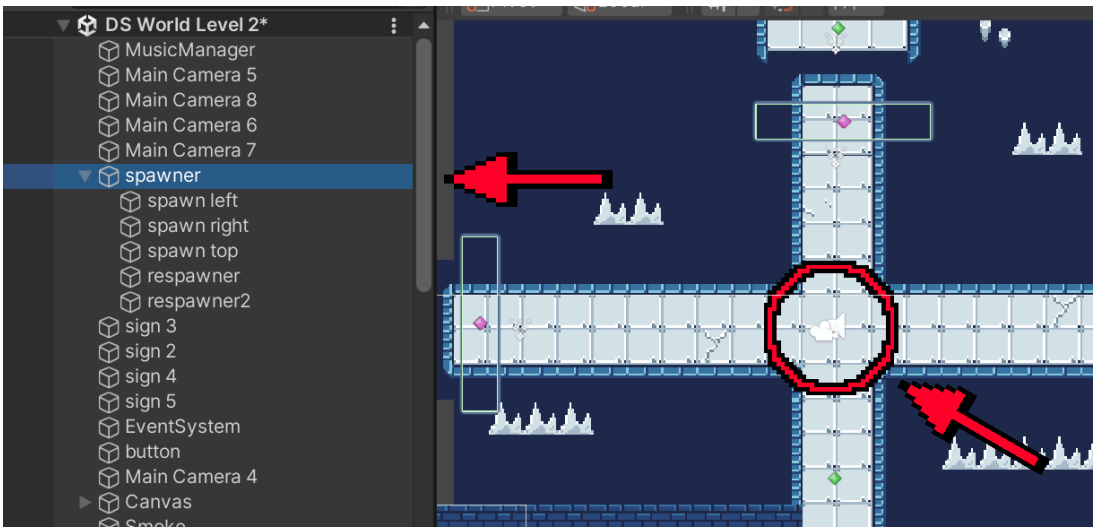

Figure 111: Image of `Spawner` script attached to a spawner `GameObject` in DS World Level 2. This script also exists DS World Level 4.

Function description

- `Start`: Disables last used camera.
- `Update`: Check if the `finished` boolean has been triggered from the `LabyrinthDoor` script.
- `FixedUpdate`: move the player up by a fixed amount (simulates the player pressing a directional input).
- `OnTriggerEnter2D`: Check if the player collides with `GameObject` and if this is the case teleport the player to a specified location, update the animation sprites and disable the player's movement input.
- `MovePlayer`: Wait 0.75 seconds, then re-enable the player's movement input.

6.2.2.63. `SpecialTimer`

Figure 112: Image of `SpecialTimer` script attached to a `timer` `GameObject` in DS World Level 5. The `timer` `GameObject` is not a physical object.

Function description

- `Update`: Update the timer by counting down.
- `UpdateTimer`: Update the UI for the timer.
- `OnTriggerEnter2D`: Check if the player collides with `GameObject` and if this is the case, start the timer countdown.
- `Restart`: Wait for 5 seconds, then reload the current scene.
- `PlayMusic`: Wait for 3 seconds, then change the music and play the music.

6.2.2.64. `Teleporter`

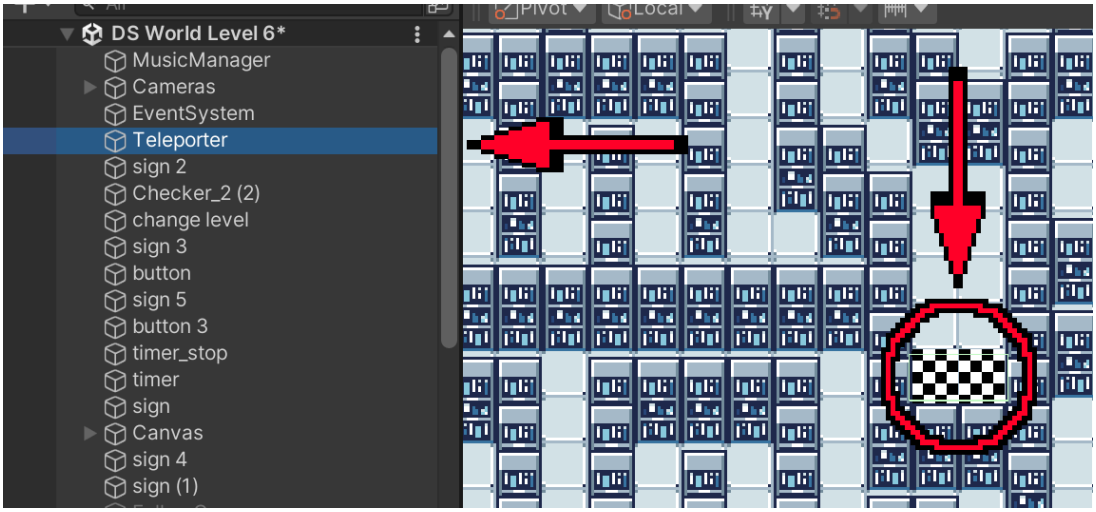

Figure 113: Image of `Teleporter` script attached to a `Teleporter` `GameObject` in DS World Level 6. The `teleporter` `GameObject` is not a physical object, but has a collision trigger attached that activate the script's function.

Function description

- `OnTriggerEnter2D`: Check if player collides with `GameObject` and if this is the case, move the player to a specified location in the scene and disable the hole `GameObject`.

6.2.2.65. `Timer`

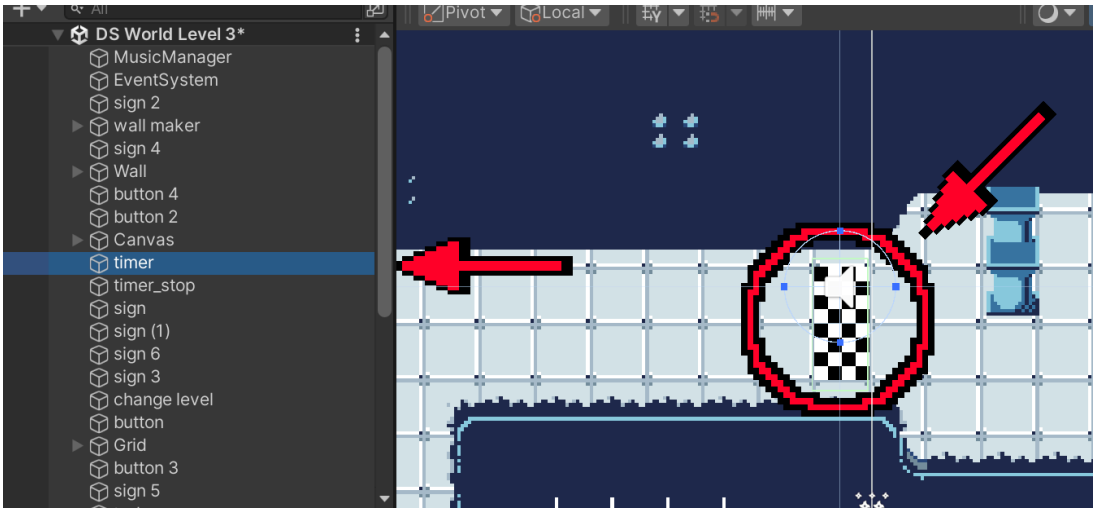

Figure 114: Image of Timer script attached to a timer `GameObject` in DS World Level 3. This script is also used in DS World Level 6.

Function description

- `Update`: Update the timer by counting down.
- `UpdateTimer`: Update the UI for the timer.
- `OnTriggerEnter2D`: Check if the player collides with `GameObject` and if this is the case, start the timer countdown.
- `Restart`: Wait for 5 seconds, then reload the current scene.
- `PlayMusic`: Wait for 3 seconds, then change the music and play the music.

6.2.2.66. `TimerStop`

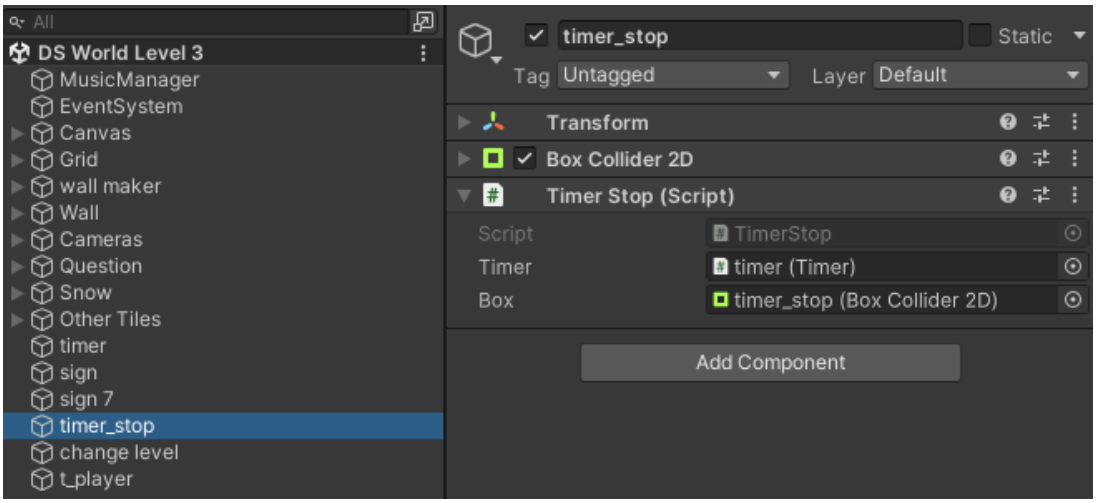

Figure 115: Image of `TimerStop` script attached to a `timer_stop` `GameObject` in DS World Level 3. The `timer_stop` `GameObject` is not a physical object, and is also used in DS World Level 6.

Function description

- `OnTriggerEnter2D`: Check if player collides with `GameObject` and if this is the case, stop the timer from running.

6.2.2.67. `TimerStopSpecial`

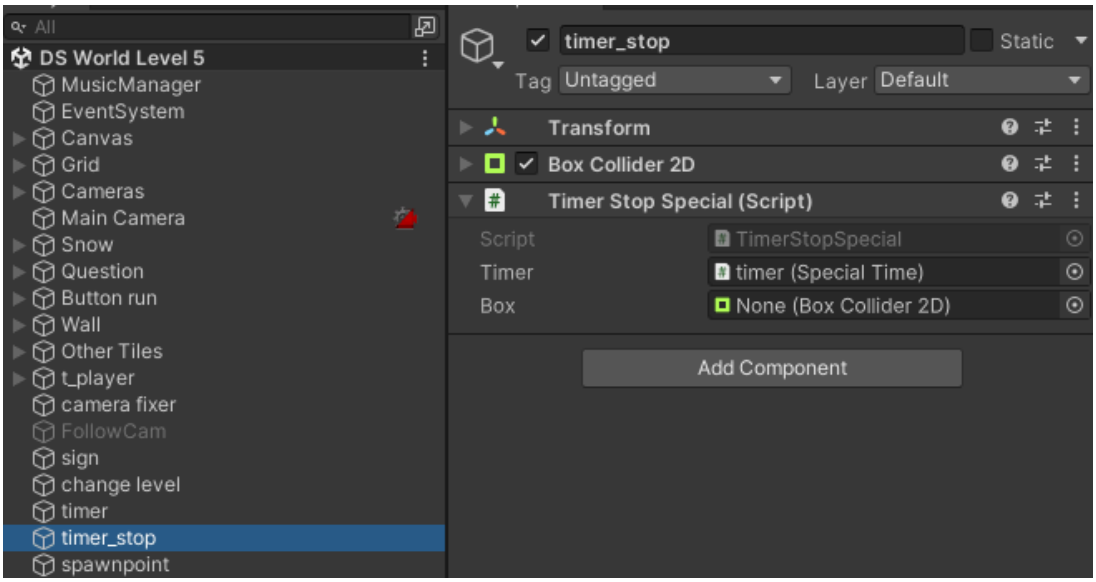

Figure 116: Image of `TimerStopSpecial` script attached to a `timer_stop` `GameObject` in DS World Level 5. The `timer_stop` `GameObject` is not a physical object.

Function description

- `OnTriggerEnter2D`: Check if player collides with `GameObject` and if this is the case, stop the timer from running.

6.2.2.68. `WallMaker`

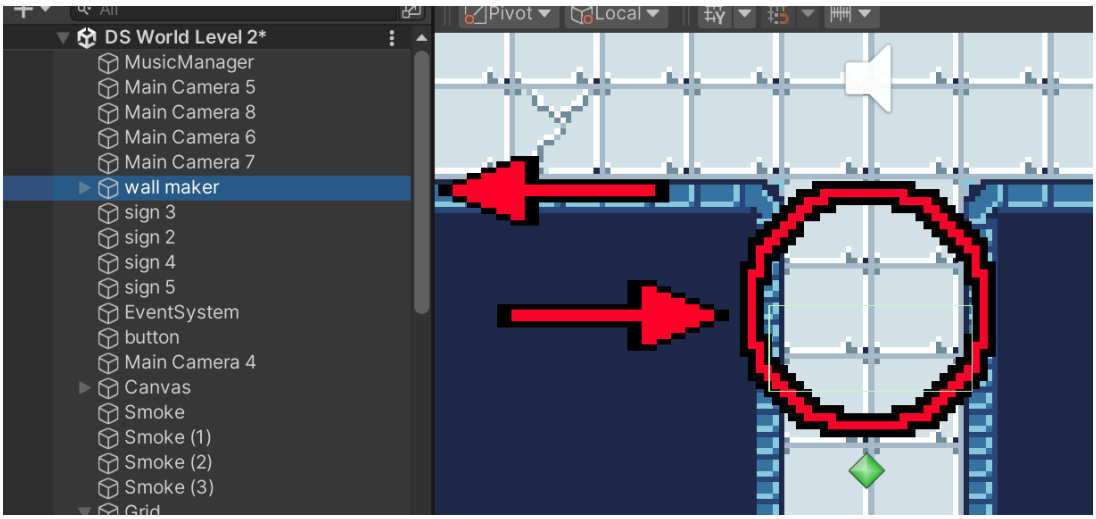

Figure 117: Image of `WallMaker` script attached to a `wall maker` `GameObject` in DS World Level 2. The `wall maker` `GameObject` is not a physical object, but has a collision trigger attached that activates the script's function.

Function description

- `OnTriggerEnter2D`: Check if the player collides with `GameObject` and if this is the case, enable a `Child` `GameObject`, which has a collision `GameObject`.

6.2.2.69. `AddEscape`

Located in puzzle 3 of AI world, attached to the `AddEscape` `GameObject`.

Function description

- `Start`: Initialises objects to add as an array which is made up of all the children of an object specified.
- `Update`: Checks if the objects should start being added based on the public boolean variable.
- `StartEscape`: Adds the desired objects, and removes the other objects specified.

6.2.2.70. `Portals`

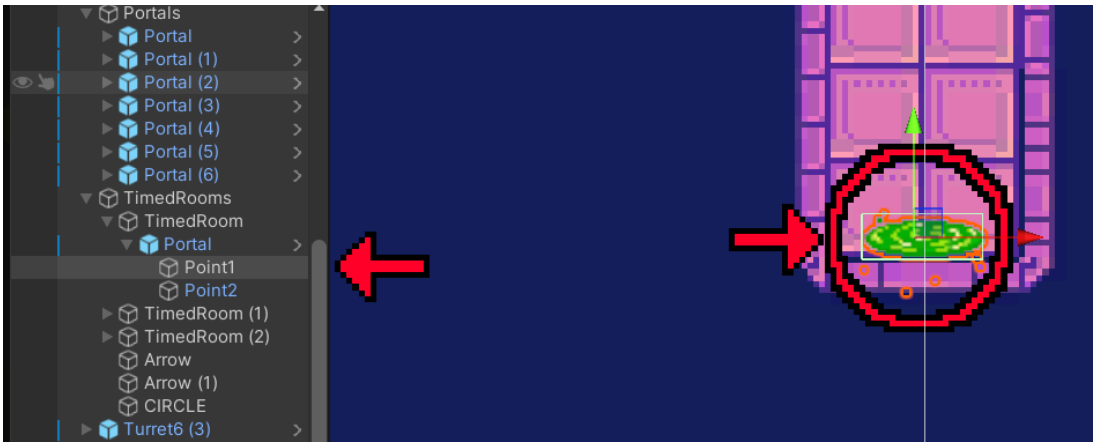

Figure 118: Image of `AIPortal` script attached to a Portal `GameObject` in Puzzle-4.

Function description

- `Start`: Initialise parent from `GameObject`, initialise player object by getting its tag.
- `OnTriggerEnter2D`: If the parent object is not on cooldown, and if the object which collided is the player, teleports the player to the next portal.

6.2.2.71. `AudioManager`

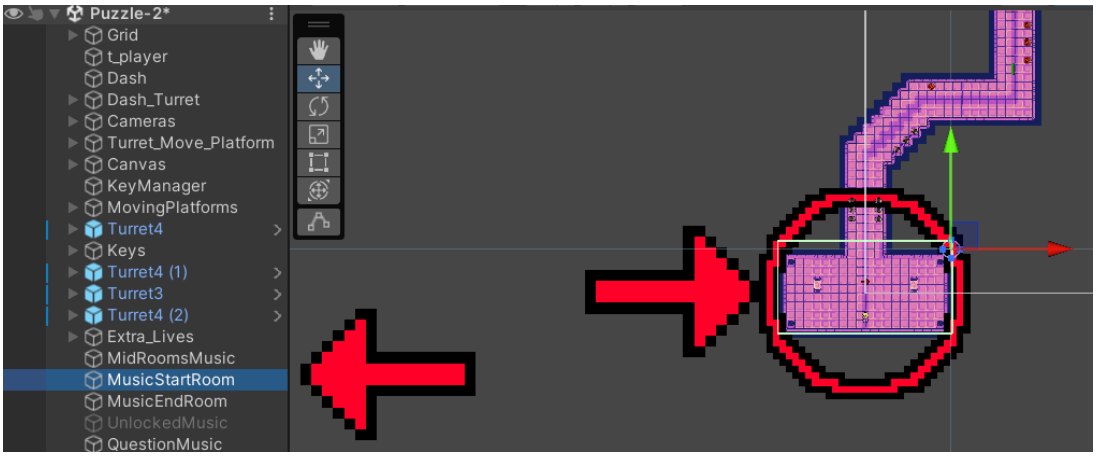

Figure 119: Image of `AudioManager` script attached to a `MusicStartRoom` `GameObject` in Puzzle-2.

Function description

- `OnTriggerEnter2D`: Plays background music specified.
- `OnTriggerExit2D`: Stops background music played when the player enters the box.

6.2.2.72. `BasicEnemyTurret`

Figure 120: Image of `BasicEnemyTurret` script attached to a Turret `GameObject` in Puzzle-4.

Function description

- `Start`: Calls the `Set Start Rotation` and `Get Spawn Points` methods from the abstract parent class.
- `FixedUpdate`: Calls the `RotateTurret` method from the abstract parent class.
- `ShootHandler`: Alternates through each state of the machine and calls the `Fire` method whilst the object is in a state (waits specified reloading time for this state between each shot). Calls the `Coroutine StartTimer` to change state. Changes the direction of the turret if the boolean `statesChangesDirection` is set to true, and calls at the end to move on to the next state.
- `StartTimer`: Waits an amount of time specified for the current state and then changes to the next state based on `nextState` dictionary.
- `GetVelocity`: Uses a `switch` statement to get the velocity of the current state.
- `Fire`: Uses a `switch` statement to get the appropriate firing function of the current state.

6.2.2.73. `Beam`

Figure 121: Image of `Beam` script attached to a `Beam GameObject` in Puzzle-4.

Function description

- `Start`: Calls the `Coroutine DestroyTimer` and `ManagePhase`, initialises the `line renderer` and `player` variables.
- `Update`: Calls the shoot function and updates the `GameObject`'s position and rotation based on the spawn point's position and rotation.
- `DestroyTimer`: Waits a specified time and then destroys the `GameObject`.

- `Shoot`: Uses Unity's 2D Physics' `Raycast` method to get the stopping point of the laser, and the box which collided with it. Calls the player's `DamagePlayer` method if the raycast hits the player (whilst the beam is not in its start phase), it then draws the beam using the position of the raycast hit as the end position.
- `DrawBeam`: Takes in two arguments, `startPosition` and `endPosition` and uses this to draw the beam using the `line renderer` of the beam.
- `ManagePhase`: Changes the colour and phase of the beam after a specified amount of time.

6.2.2.74. `Bullet`

Figure 122: Image of `Bullet` script attached to a fast_bullet `GameObject` in Puzzle-4.

Function description

- `Update`: Moves the bullet with specified velocity towards its current direction.
- `OnTriggerEnterCollision2D`: Calls the player's `DamagePlayer` function if the object it collided with has the `Player` tag and the player was not dashing. Destroys the `GameObject` if the object collided with has the "`Collide`" tag.

6.2.2.75. `Dash`

Function description

- `Update`: Calls the `StartDash` function if the dash is not on cooldown and the player presses the space key.
- `StartDash`: Moves the player in its current direction for a set amount of time and puts the dash on a cooldown.

6.2.2.76. `DashItem`

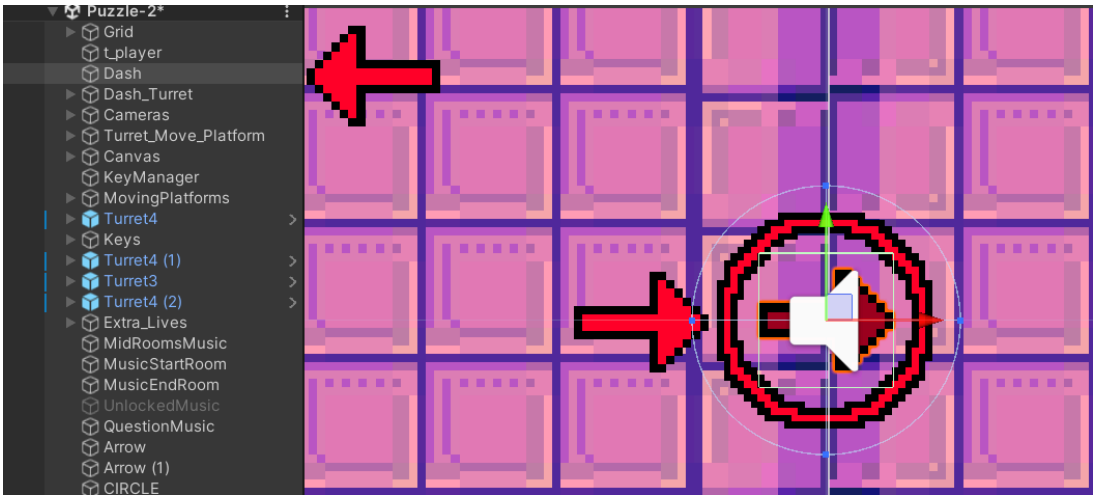

Figure 123: Image of `DashItem` script attached to a `Dash` `GameObject` in Puzzle-2.

Function description

- `Update`: Pauses current background music if any background music is playing.
- `OnTriggerEnter2D`: Sets the `canDash` value of the `Dash` object attached to the player's to true.
- `PlayAudio`: Plays the attached sound effect and disables the object's collision box and sprite renderer. Waits for a set amount of time and then stops the sound effect and resumes the background music.

6.2.2.77. `EscapeButton`

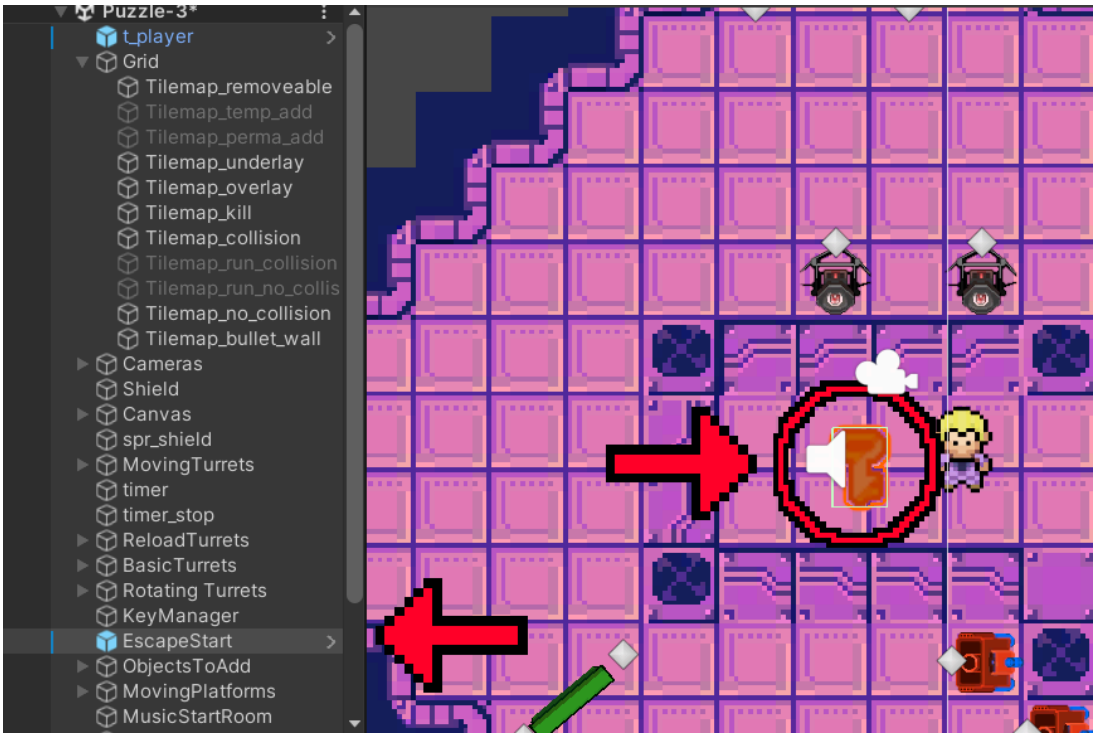

Figure 124: "Image of `EscapeButton` script attached to a `EscapeStart` `GameObject` in Puzzle-3."

Function description

- `OnTriggerEnter2D`: If the collided object has the "`Player`", sets the `key manager`'s `escapeStart` boolean to true and destroys the `GameObject`.

6.2.2.78. `Hearts`

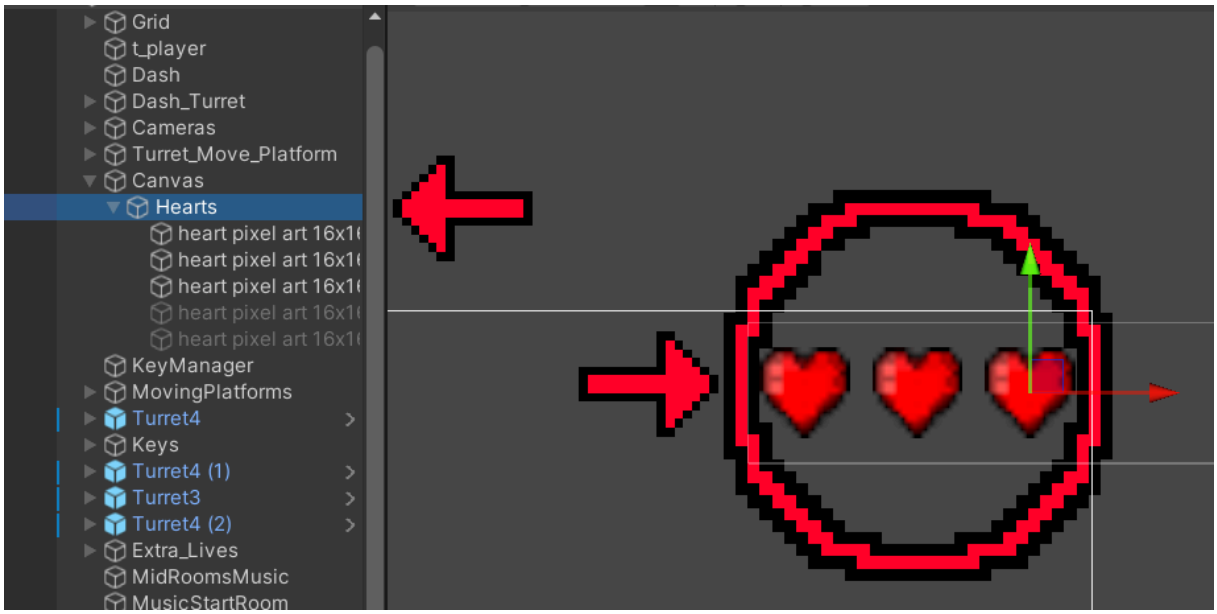

Figure 125: Image of `Hearts` script attached to a Hearts `GameObject` in Puzzle-2.

Function description

- `Start`: Initialises the `player` variable, and the `heart` variables by getting the child objects of the object.
- `Update`: Gets the number of lives of the player and calls the `ShowHearts` function if the number of lives has changed.
- `ShowHearts`: Activates the hearts on the canvas to active if their index in the hearts array is less than the number of hearts, and deactivates the rest.

6.2.2.79. `HomingDrone`

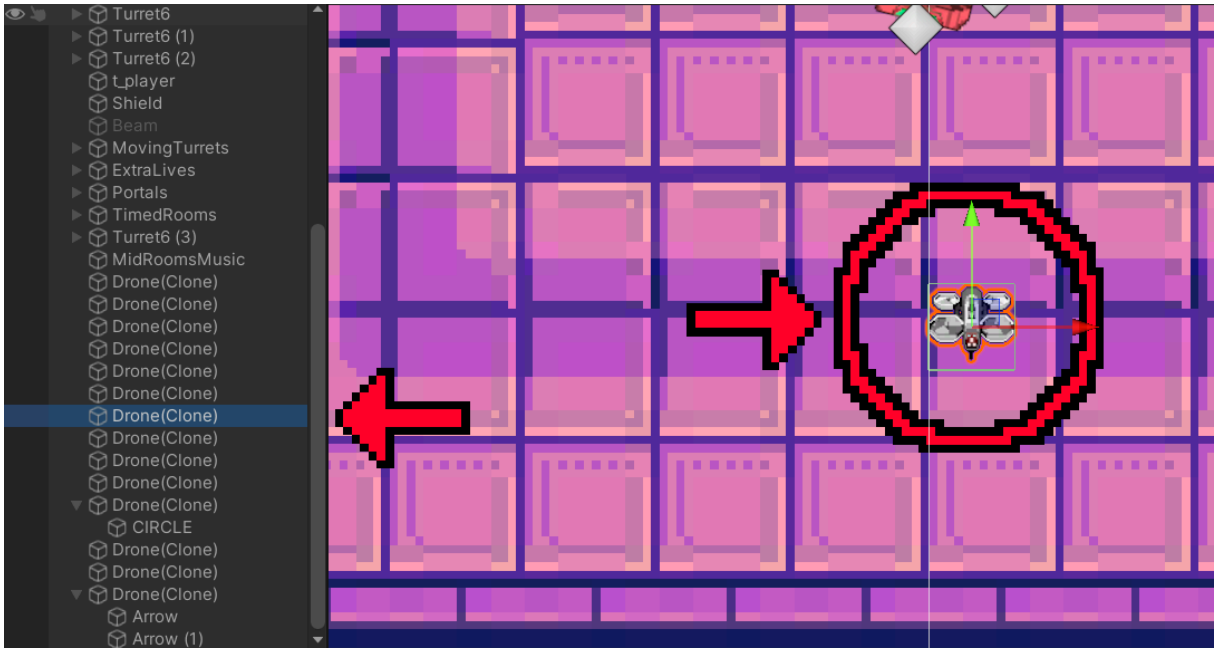

Figure 126: Image of `HomingDrone` script attached to a `Drone` `GameObject` in Puzzle-4.

Function description

- `Start`: Initialises the player object variable and starts the coroutines `DeathTimer` and `StartPhaseTimer`
- `StartPhaseTimer`: Waits a specified amount of time and then sets the `homingPhase` boolean to true.
- `DestroyTimer`: Waits a specified amount of time and then starts the animation and starts the `EndDrone` coroutine.
- `FixedUpdate`: If during `homingPhase`, move the object towards the player, else moves the object in its current direction.
- `Update`: If the object is not in the ending animation and the player is close enough, call the player's `DamagePlayer` method, starts the end animation and calls the `EndDrone` coroutine.
- `OnTriggerEnter2D`: If the collided object has the "`Collide`" tag, starts the `EndDrone` Coroutine.
- `EndDrone`: Waits a specified amount of time and then destroys the object.

6.2.2.80. `KeyManager`

Function description

- `Update`: Checks the number of keys, if this is 0 activates and deactivates specified objects and (if specified) calls the `StartMusic` coroutine.
- `StartMusic`: Deactivates/activates the specified `GameObjects`, and changes the audio playing.

6.2.2.81. `Keys`

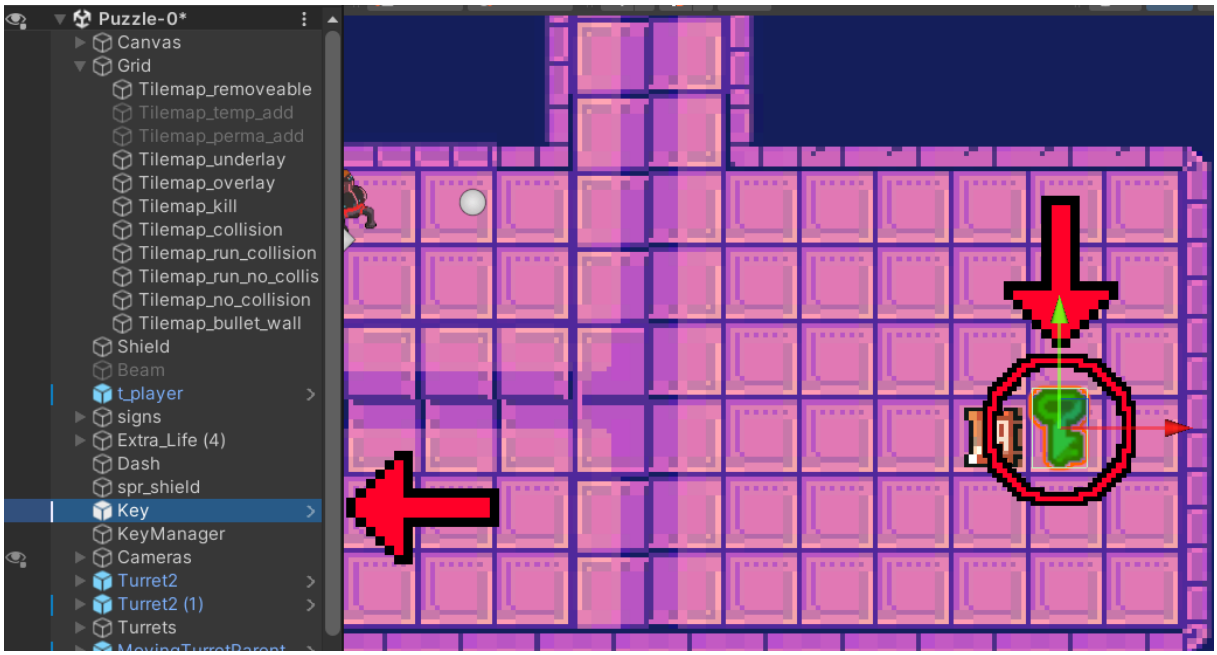

Figure 127: Image of `keys` script attached to a `key` `GameObject` in Puzzle-4.

Function description

- `OnTriggerEnter2D`: If the object that collided has the "`Player`" tag, reduce the variable `keyNum` of the `KeyManager` by 1, and destroy the object.

6.2.2.82. `KillPlayer`

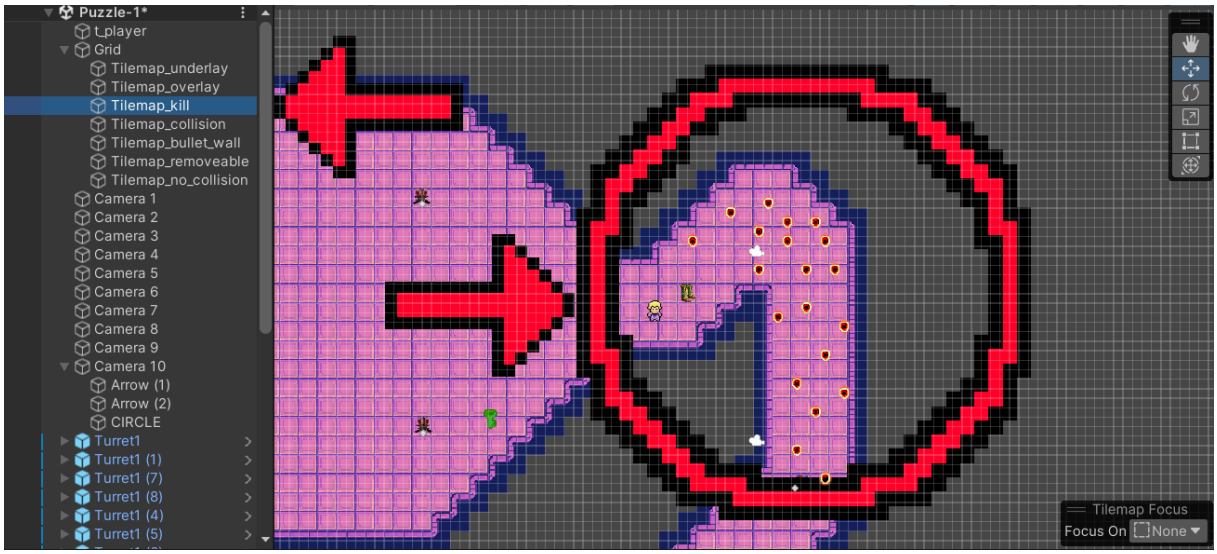

Figure 128: Image of obstacles (highlighted) which the player must avoid.

Function description

- `OnTriggerEnter2D`: If the collided object has the player tag, calls the `DamagePlayer` method of the player.

6.2.2.83. `Lives`

Function description

- `Start`: Initialise the `sprite` and `anim` variables by getting them from the object component.
- `Update`: Checks the number of lives of the player and reloads the scene if this is 0.
- `DamagePlayer`: If the player is not invincible, reduces the number of lives by 1, sets invincibility to true and calls the `Flicker` and `Invincible Timer` coroutines.
- `Flicker`: Turns the animation and sprite on and off at a specified rate whilst the player is invincible.
- `InvincibleTimer`: Waits a set number of seconds and then turns the invincibility off.
- `OnTriggerEnter2D`: If the collided object has the "`Heart`" tag, increase the number of lives by 1 and destroy the object which collided (not the object the script is called to).

6.2.2.84. `MovingPlatform`

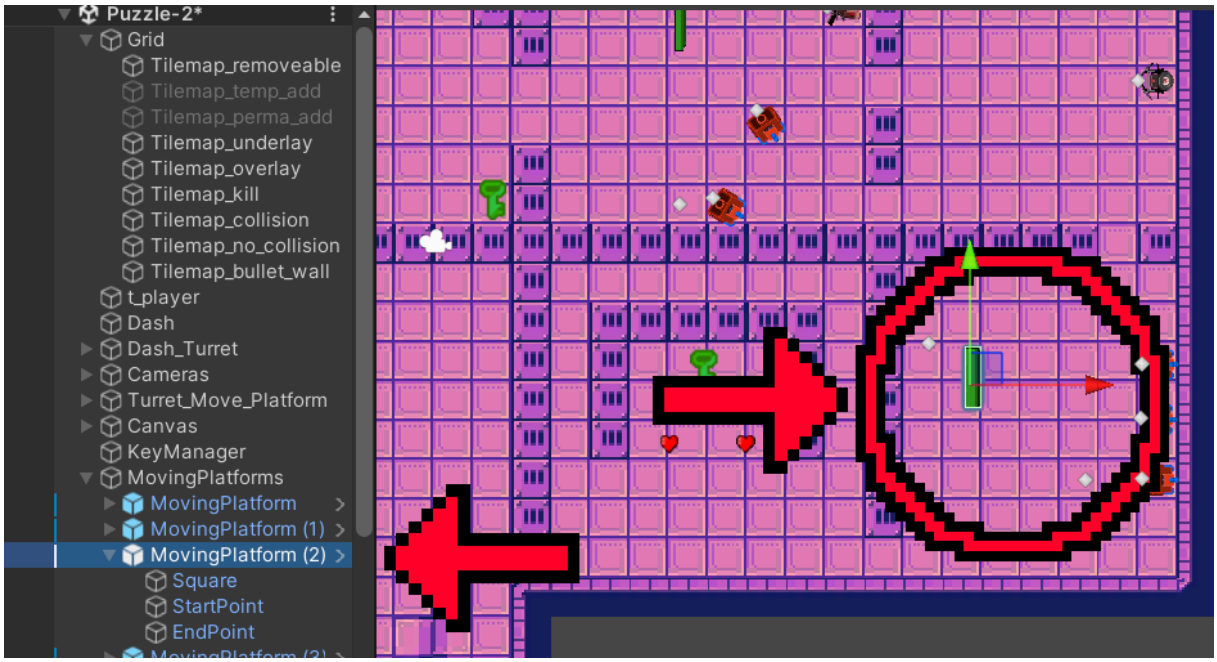

Figure 129: Image of `MovingPlatform` script attached to a platform `GameObject` with two points under the same parent object in Puzzle-4.

Function description

- `Start`: Gets the `startPoint` and `endPoint` as the positions of the specified start and end `GameObjects`, sets the velocity variable to the specified velocity.
- `FixedUpdate`: Change the position of the object by the set velocity. Checks if the object is within a specified distance from the end/start (depending on `movingDown` boolean) and if so multiply the velocity by −1 to reverse its direction.

6.2.2.85. `PlayerMovement2`

The `player` `GameObject` holds this script in every level in the AI World.

Function description

- `Update`: Updates the player's position based on the directional input keys pressed, and updates the player's animation sprites based on this movement. It also disables the player's movement when on ice.
- `FixedUpdate`: Physically moves the player rather than just updating the movement values.
- `Flip`: Flips the player sprite when moving left.
- `OnTriggerEnter2D`: Check if the player collides with `GameObject` of type ice and if this is the case, disables the movement input for some time.
- `OnTriggerExit2D`: Check if the player leaves the collision range of the ice `GameObjects` and if this is the case, re-enable the player's movement input.

6.2.2.86. `PortalParent`

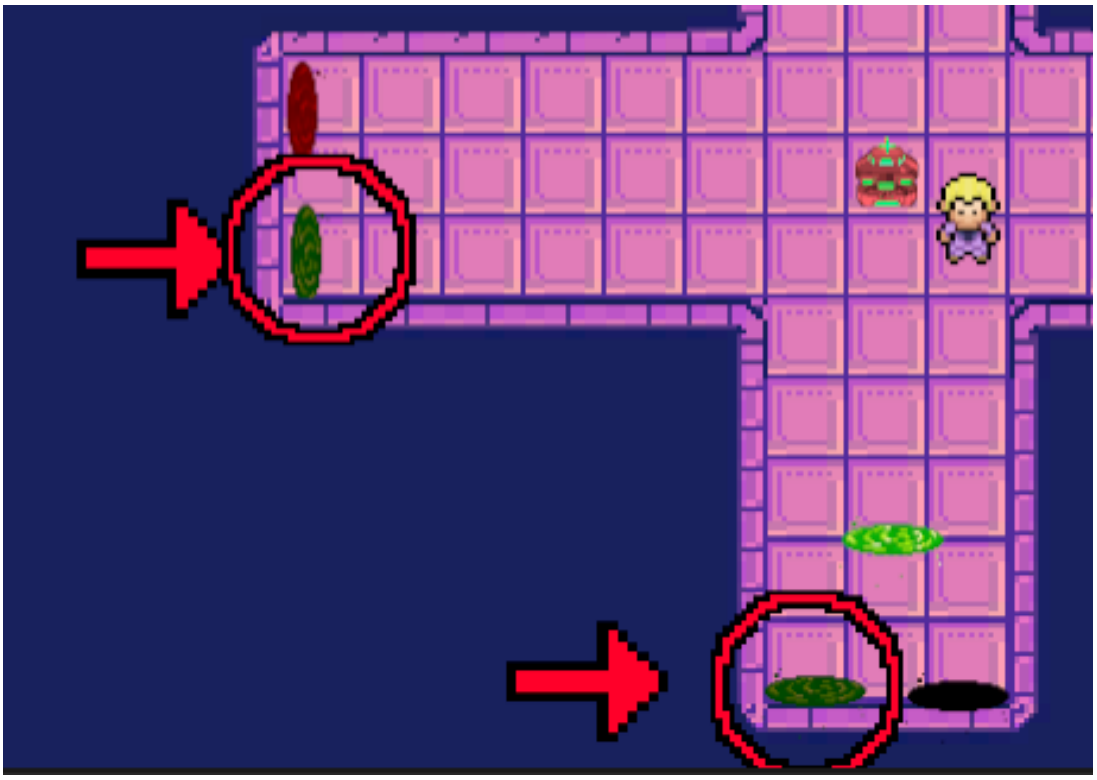

Figure 130: Image showing the portal parent and the points of the portal (green portals)

Function description

- `Start`: Gets the number of portals and initialises the portals array.
- `GetNextPortal`: Starts the teleport cooldown and returns the next portal in the array (loops back around on the last index).
- `StartCooldown`: Sets the `cooldown` boolean to true, then waits for a specified amount of time before switching the `cooldown` boolean back to false. If the portal should disable on use, call the `DisableAll` function.
- `DisableAll`: De-actives all the portals in the portal array.

6.2.2.87. `Shield`

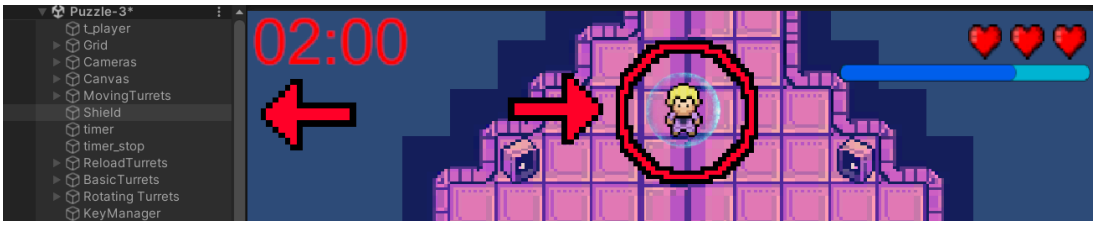

Figure 131: Image showing the player using the shield ability, follows the player around

Function description

- `Start`: Initialises the `player`, `shieldBar` (the slider on the canvas), `shieldCollider` and `sprite` variables, sets the shield to ignore collision with the player.
- `Update`: If the `canShield` boolean is true and the player holds down the "`Fire3`" button, activates the collider and sprite of the shield, else, disables the sprite and collider of the shield. Updates the shield's position so that is equal to that of the player.

- `FixedUpdate`: If the player `isShielding`, decrease the value of the shield bar slider by a fixed rate. If the value of the shield bar is 0, set the `canShield` boolean to false.

6.2.2.88. `ShieldItem`

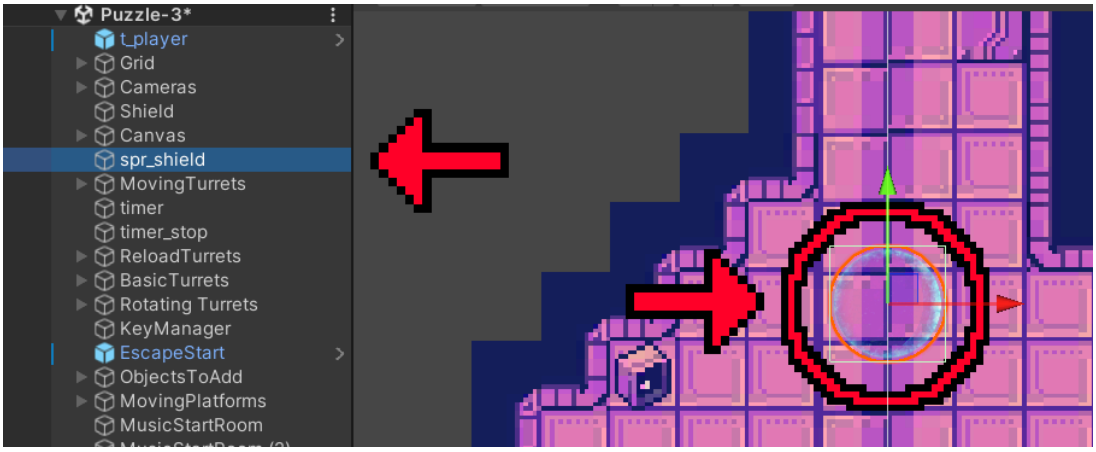

Figure 132: Image showing the item which gives the player the ability to shield.

Function description

- `OnTriggerEnter2D`: If the collided object has the player tag, set the `canShield` boolean of the shield to true, activate the shield and destroy the object attached to the script.

6.2.2.89. `SpeedBoost`

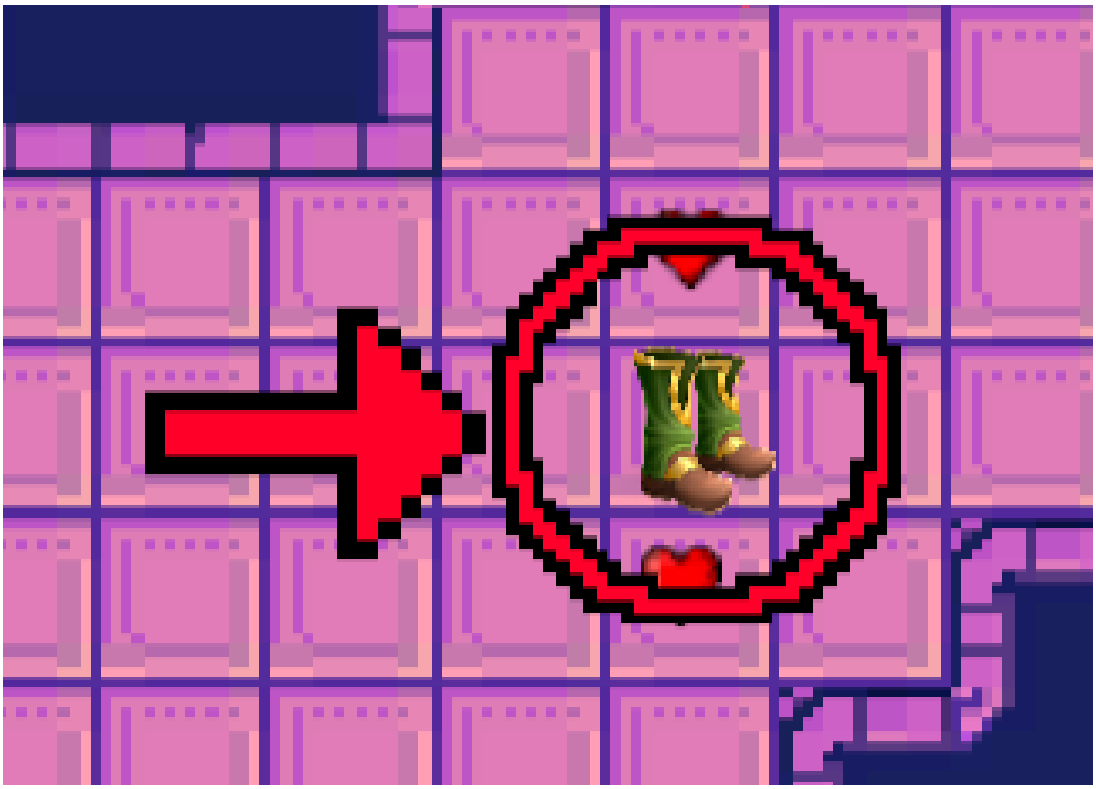

Figure 133: Image showing the item which increases the player's movement speed

Function description

- `OnTriggerEnter2D`: If the collided object has the "`player`" tag, change the player's movement speed to 10 (double normal movement speed) and destroy the object the script is attached to.

6.2.2.90. `TimedRoom`

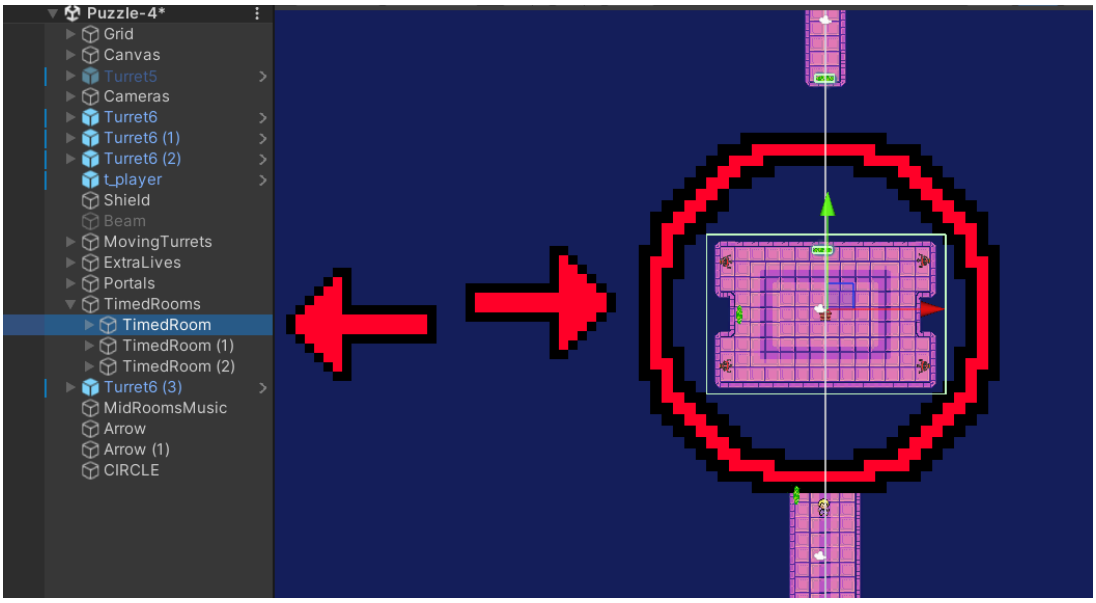

Figure 134: Image showing the room which the player must survive in for a certain amount of time to progress, with the bar at the bottom.

Function description

- `Start`: Initialises portal object and sets the `timeBar` value to 0, deactivates the portal.
- `FixedUpdate`: If the timer is on, decreases the value of the `timeBar`. If this value is 0, activate the portal and deactivate the parent of the slider, resets the value of the `timeBar`.
- `OnTriggerEnter2D`: If the collided object has the "`Player`" tag, turn the timer on and enable the slider as well as the parent object of the slider.

6.2.2.91. `Timer2`

The `timer GameObject` has the `Timer2` script attached, which is used in AI World 3.

Function description

- `Update`: Update the timer by counting down.
- `UpdateTimer`: Update the UI for the timer.
- `OnTriggerEnter2D`: Check if the player collides with `GameObject` and if this is the case, start the timer countdown.
- `Restart`: Wait for 5 seconds, then reload the current scene.
- `PlayMusic`: Wait for 3 seconds, then turn off the timer music.

6.2.2.92. `TimerStop2`

Figure 135: Image showing where the `GameObject` which contains the timerstop2 script.

Function description

- `OnTriggerEnter2D`: If the collided object has the "`Player`" tag, sets the `stoptimer` boolean of the timer to true and enables collider of the object the script is attached to.

6.2.2.93. `Turret`

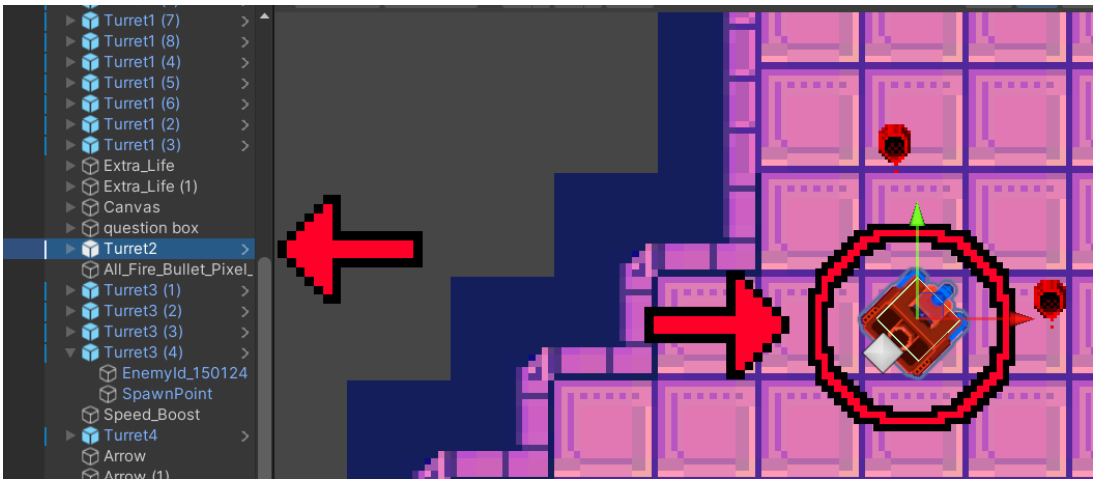

Figure 136: Image showing the basic turrets

Function description

- `Start`: Initialises the start angle.
- `FixedUpdate`: Rotates the object by specified speed, inverses direction if close to start or end angle.
- `Fire`: Takes in a `spawn GameObject` as a parameter and instantiates a specified prefab at the `spawn`'s position and rotation. Sets the instantiated object's speed to a specified speed.

- `FireBullets`: Waits for a specified number of seconds and then calls the Fire method at a specified rate, after which it reloads for a specified number of seconds. If the turret is meant to shoot in both directions, calls the Fire method twice for each spawn point.

6.2.2.94. `TurretStateMachine`

Function description

- `SetStartPoint`: Initialise the starting angle.
- `GetSpawnPoints`: Makes an array with all the children objects of the object which have "`Point`" in their name.
- `MakeEnemy`: For all spawn points with "`All`" or "`Enemies`" in their name, instantiate an object (which must have the `homingDrone` class) with the spawn point's position and rotation, and set the speed of this object to the speed specified.
- `ShootScatter`: For all spawn points with "`All`" or "`Enemies`" in their name, instantiate a `bullet` with a specified speed a specified rate.
- `ShootStraight`: For all spawn points with "`All`" or "`Enemies`" in their name, instantiate a `bullet` with a specified speed a specified number of times at a specified rate.
- `ShootBeam`: For all spawn points with "`Beam`" in their name, instantiate a `beam` which lasts for a specified time.
- `GetPrefab`: Returns the prefab corresponding to the current state.

6.3. System maintenance

In this section, we will go over the main technical aspects of our game that might require maintenance. We also cover how to work with them, which will be useful for Section 6.4.

6.3.1. New Worlds

To create a new world, the following steps should be carried out:

1. **Obtain a tileset:** This can either be found online (given that there is a licence) or it can be hand drawn. After importing the tileset change the sprite rendering mode from "single" to "multiple" and then autoslice the tiles into even cube shapes. Then create a tilepallet and add all the newly imported tiles to it. Finally, use the brush tool to create a suitable level for the player. Remember to switch between tile plates that have collision enabled and ones that don't to create an environment the player can roam around but cannot exit the desired play space.
2. **Get IBM Questions:** A new IBM Skills Build course should be chosen (as the game should cover a variety of topics). Then, gather the questions, append them onto the central questions CSV file located inside the "Resources" folder in the appropriate format and run the import questions functionality in the Unity editor (see Section 6.2.2.15 for more details).
3. **Copy core game mechanics:** Reuse key parts of our game, which should include the player object, the template for the boss fight and the cameras (unless you wish to have a different system for how the player stays on screen). Note that other objects/scripts can be copied/used as well if the world to be created has similar features to one already created. For example, the AI World contains a dash script which can be used if another world needs to have a dashing ability.
4. **Add new puzzle mechanics:** Implement a new puzzle mechanic/main world concept, such as turrets in AI World or timer challenges in DS World.
5. **Add Boss fight:** A level which leads up to the boss fight should first be added. After this, the actual combat scene copied from previous worlds should be added, but with some changes to the sprites as well as images to match the world. After setting the questions up in step 2, make sure the `QuestionManager` in the scene uses questions from the right folder (see Section 6.2.2.27 for more details).
6. **Hub World linking:** Link the newly created world to the Hub world by adding a room with a magic circle in it, and set the world to be teleported to by that magic circle to be the newly created world. Add a door to the Hub World in order to be able to enter that magic circle room.
7. **Build tests:** During the development of this world, unit tests should be added incrementally, followed by integration tests. Once the world is finished, system tests can be added, if those pass, you can move on to user acceptance tests to get playtesting feedback.

This general structure holds true for the existing worlds that we currently have. Therefore, if there is an issue with a world, check that each bullet point has been covered properly. A lot of the bugs will be collision issues, which can be fixed inside Scene view in the Unity Editor by moving the tile objects around.

In Section 6.4, we have several other worlds we only started development on, and ideas that were still in its conceptual stage of development.

6.3.2. Settings Menu

The Settings Menu has been created with future implementability in mind: Updates to the menu are easy to implement and require little to no changes to the general infrastructure of the settings menu.

All the settings menu controls are controlled by the `GM`, which updates the user's settings and stores the values for the save mechanic. As such, it has been designed linearly, with each setting being its own

function in the `GM` script. Thus, new settings can be easily added in future developments by creating another function for the new addition.

To create a new Setting in the Menu, the following steps should be carried out:

1. **Add a UI Button:** Inside the Unity Editor's setting menu scene, add a UI Button component. Move and align this to the desired location.
2. **Name Button:** Change the name to correspond with the new settings feature.
3. **`PauseMenu` script:** Create a corresponding function in the `PauseMenu` script.
4. **Link to `GM` script:** Link the newly created function to the `GM` script.
5. **Add `OnClick` function:** Add the newly created function to the `OnClick` function of the new UI button in the unity editor.

If a button does not perform the functionality assigned to it, first check that the object has the appropriate script attached using the Unity Editor Inspect mode. Then, if the script is present, then it is most likely an issue with the logic of the code.

6.3.3. Combat System

To add or fix more features in the combat system, we must first understand the state design pattern and finite state machines [6].

A FSM is an abstract model that holds a finite number of states and their respective transitions, while also keeping track of the current state of the system during its execution. This is useful for game development as lots of in-game systems, such as enemy actions or combat, have behaviours that can be directly mapped to a FSM.

The state design pattern exists to convert the FSM concept to code, allowing one to naturally and cleanly implement systems. This in turn keeps the code base clean and helps with extensibility, which is particularly useful in case the client wants more features added.

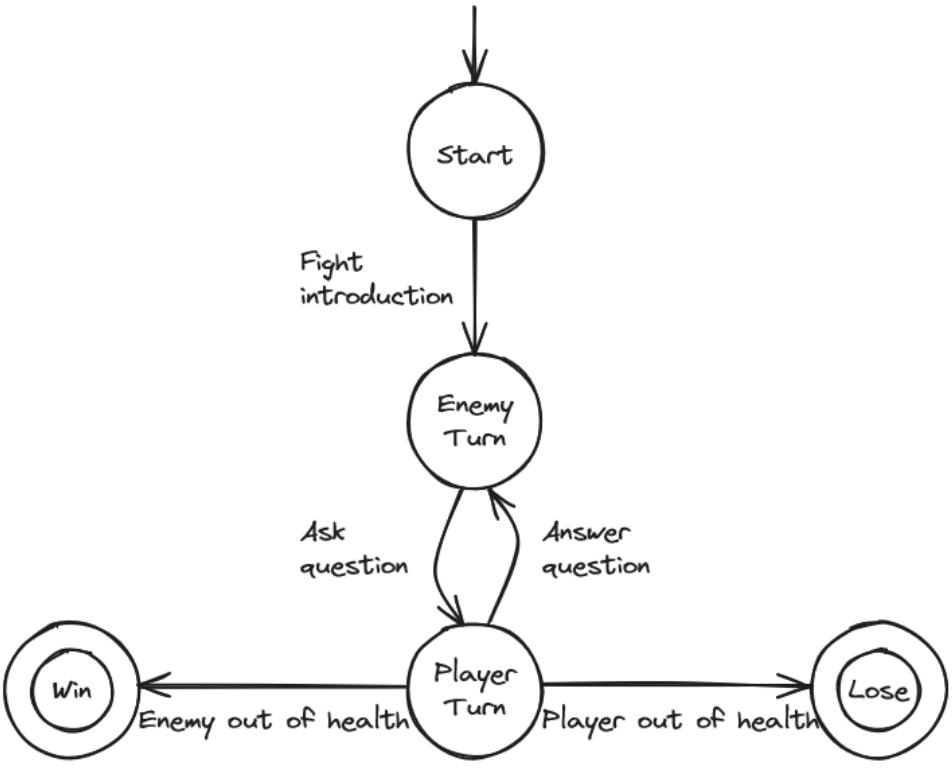

Figure 137: Image of the in-game combat finite state machine.

Instead of using `switch` statements, `enums` and lots of conditional logic branches, all components are separated into different classes which all inherit from an abstract `State` class the `start` and `answer` functions (see Section 6.2.2.29).

While states handles internal behaviours of the system, we need a larger `StateMachine` (see Section 6.2.2.30) object which maintains the current state, and handles all the transitions from there. Comprehensive knowledge of this class is not necessary for adding new combat states, but it is nonetheless important to know.

Armed with this knowledge, we can finally talk about how to add a new state. The steps to take are as following:

1. **Create a new state class:** This class should inherit from the abstract `State` class, and include the `BattleSystem` object in its constructor.
2. **Override the `Start` method:** The `Start` method dictates what the state does when transitioned into. Add additional code to achieve the desired functionality.
3. **Set the outward transitions:** Decide what the next `State` the `StateMachine` should move to. This can either be an unconditional transition, or it can have some logic to decide which transition to take.

Whenever a state breaks, or has issues, first check the code of the individual states as this holds most, if not all the logic of the states. Issues with inheritance might also occur, so make sure that the entire structure is set up appropriately (with the `StateMachine` and `State` functions).

6.3.4. Working with Questions

To modify or add additional questions to the game, follow these steps:

1. **Add questions in spreadsheet:** Navigate to the "questions.csv" file inside of the Resource folder (in the Project view), and add the questions in the format: Question, Answers, World. The answers are split by semi-colons, and The first answer in this column is the correct one. Make sure that all folders for different worlds exists before importing the questions. For example, if you wanted to add another world called "JourneyCloud", a folder called "JourneyCloud" should exist inside "Resources".

	A	B	C
1	Question	Answers	World
2	When hosting a Mural collaborative sessic	FALSE;TRUE	AI
3	What is a dialog skill?	A container for the artifacts that def	AI
4	Which Dialog Nodes are not automatically	About_restaurant;Welcome;Genera	AI
5	Is it true that chatbot responses are only li	No - chatbot responses can also incl	AI
6	By setting up several values within an Enti	TRUE;FALSE	AI
7	A chatbot can be more than just a questio	TRUE;FALSE	AI
8	Which of the following best describes the	Deep Learning is a subset of Machin	AI
9	AI learning process entails human and ma	Learns by adjusting weights and bia	AI
10	Which of the following learning models is	Reinforcement learning;Supervised	AI
11	Cognitive (AI) systems are best characteri	Interact;Predict;Optimize;Converge	AI
12	For a natural language processing (NLP) sy	Innuendos and nuances used in hun	AI
13	Which of the following is a true statement	NLP is a classification problem;NLP i	AI
14	Some of the best practices used in buildin	All of the others;Introduce Your Cha	AI

Figure 138: Image of the CSV file containing a list of questions, answers, as well as topics from IBM Skills Build's various badges.

2. **Import questions into Unity:** To import the questions, place the CSV file inside the "Resources" folder, then press on "Utilities" → "Import Questions". This should have generated the questions as `ScriptableObjects` inside the appropriate world folders.

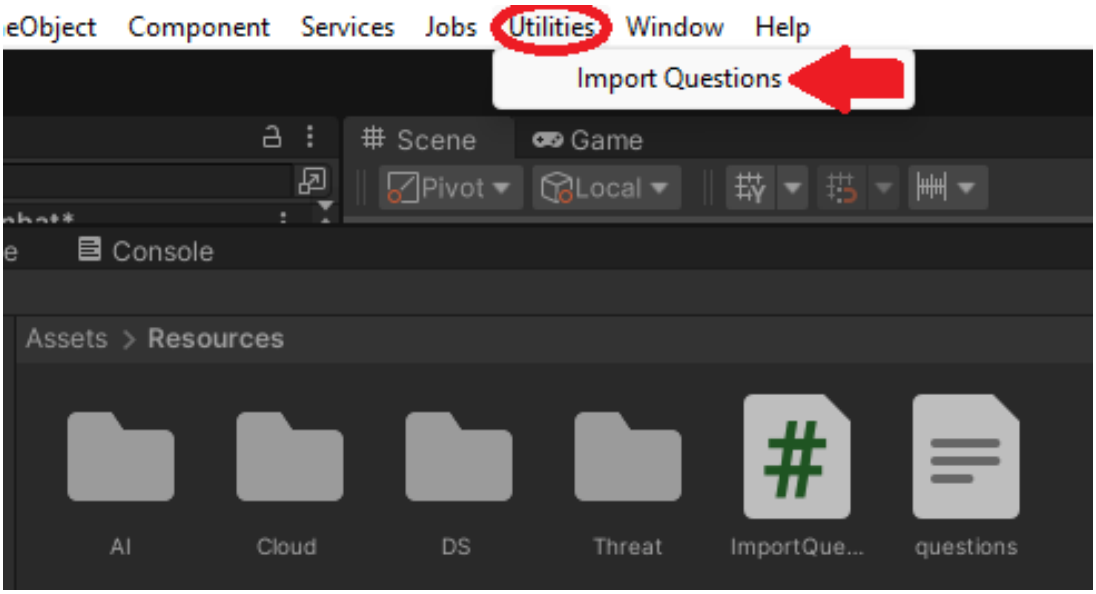

Figure 139: Image of the import question option in Unity.

3. **Update fight scene:** If the questions added are for a new world, make sure to change the `QuestionManager` to use this world name (see Section 6.2.2.27).

6.4. Future Developments

In this section, we will go over what additional features and content could be for the game. While this does not cover everything, it should at least be helpful for future developers.

6.4.1. Cloud World

The primary concept for the cloud world is to traverse a series of islands in the sky, where you can walk on clouds and grass alike. In addition to regular question puzzles,

Figure 140: Image of our concept for the Cloud World.

The main puzzle concept for this world is jumping puzzles. A jumping puzzle is a type of puzzle where there are a series of floating platforms and the player needs to jump between them. There are many variations of jumping puzzles; in games like Destiny 2 or Final Fantasy XIV, jumping puzzles are predominantly about how skilled the player is with the game's mechanics.

Figure 141: Image of Kugane Tower, Final Fantasy XIV's hardest jumping puzzle, which requires a series of complex jumps to scale the outside of the tower.

However, as our game is in 2D, the world will focus more on jumping puzzles which requires you to traverse platforms in a specific order to reach the other side. For example, take Undertale's Hotland jumping puzzle, where standing on the steam vent, will make the player jump to the platform the arrow is pointing in, with some platforms having buttons in the middle which change the active steam vents, meaning when the player lands, the steam vents may point in an unexpected direction.

Figure 142: Image of Undertale's Hotland jumping puzzle.

Another puzzle concept which we aim to implement in the cloud world is light and reflection puzzles. An example of this is in Portal 2, in which the player must redirect lasers using "Discouragement Redirection Cubes" to a destination to power a device in the room.

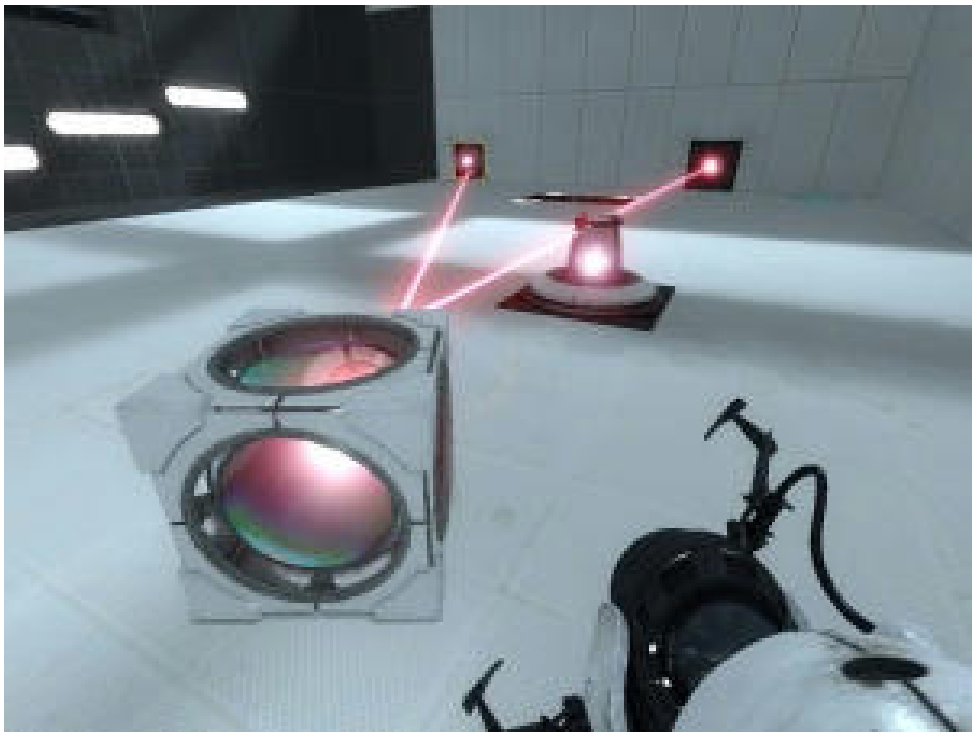

Figure 143: Image of a Discouragement Redirection Cube reflecting a laser.

We could implement this with a number of mirrors which could be rotated by a series of switches, requiring the player to rotate the mirrors into the correct position, in order to open a door or move to another sky island.

Finally, for the boss fight, one of the islands would contain a golden temple, and upon entering the player would meet the boss, beginning a boss fight on a temple in the clouds. This idea was inspired by both the Golden Temple at the end of Ma'habre in Fear and Hunger, where the player fights a powerful enemy

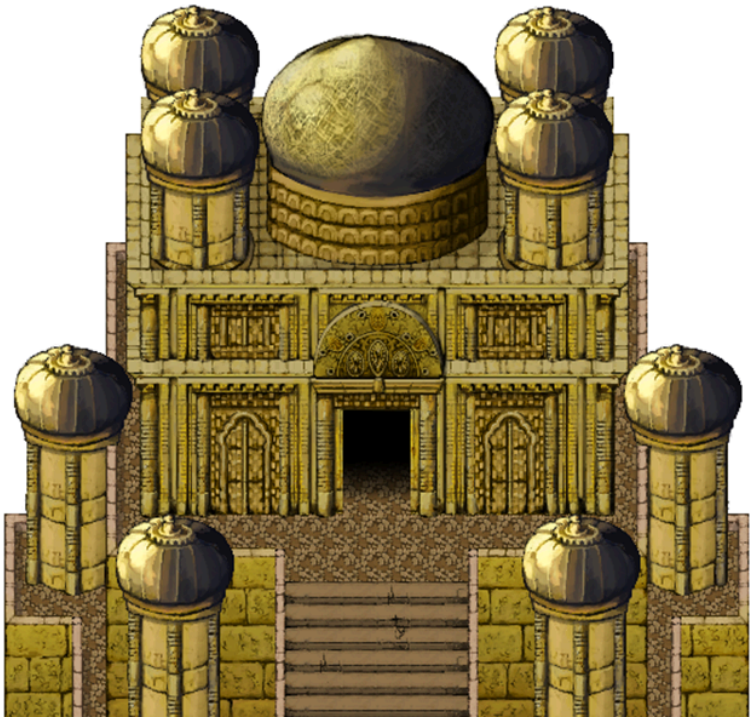

Figure 144: Image of the Ma'habre Golden Temple

6.4.2. Threat Intelligence World

The threat intelligence world is to be set inside volcanic caves, and most of the tileset would consist of rocks and lava.

The main puzzle theme for this area was intended to be block puzzles. Block puzzles come in two main forms:

Sliding block puzzles involve pushing blocks which slide in the direction the player pushes them in, until it hits a wall or another block, and then stops. For example, in the sliding block puzzle in Fear and Hunger, the player must push the block labelled 1 to the tile labelled 2. To do this the player would stand on the left of block 1, push it to the right, then standing below the block and push it upwards. Afterwards, the player must stand to the right of the block and push it left, following the blue line annotation.

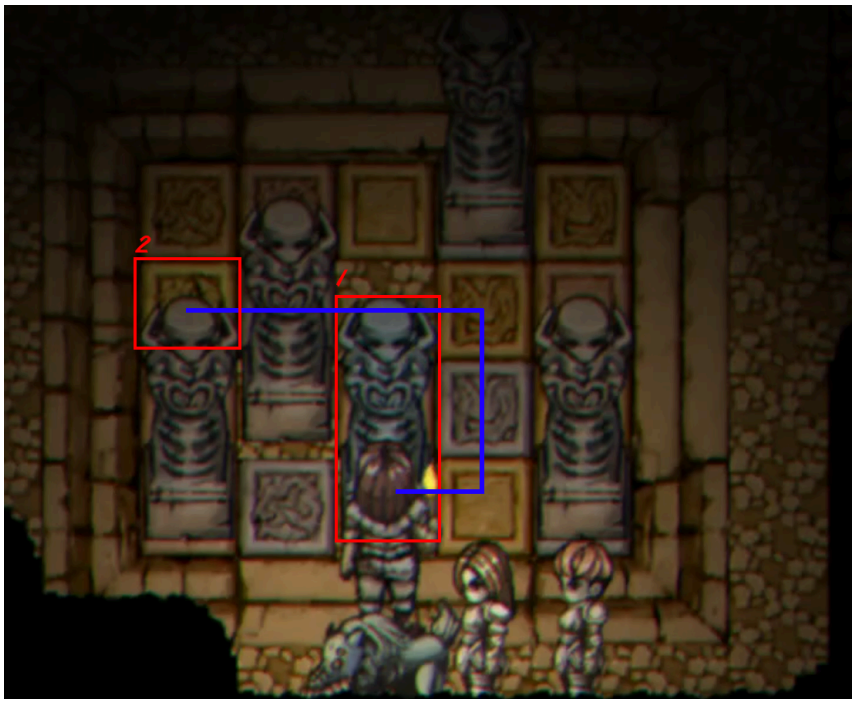

Figure 145: Image of the first block sliding puzzle in Fear and Hunger, with its solution annotated.

Pushing block puzzles are distinct from sliding block puzzles, as these blocks only move 1 tile at a time. The goal in the puzzle shown below is to get the player character to the other side of the boulders blocking the path. This is done by pushing boulders in the order shown by the numbers in the image. What makes this puzzle not possible in **sliding block style** is moving number 4 in the image, as the boulder only moves 1 tile over. It allows for the player to exit when the boulders are pushed correctly as if the boulder slid another tile along, the exit would be blocked.

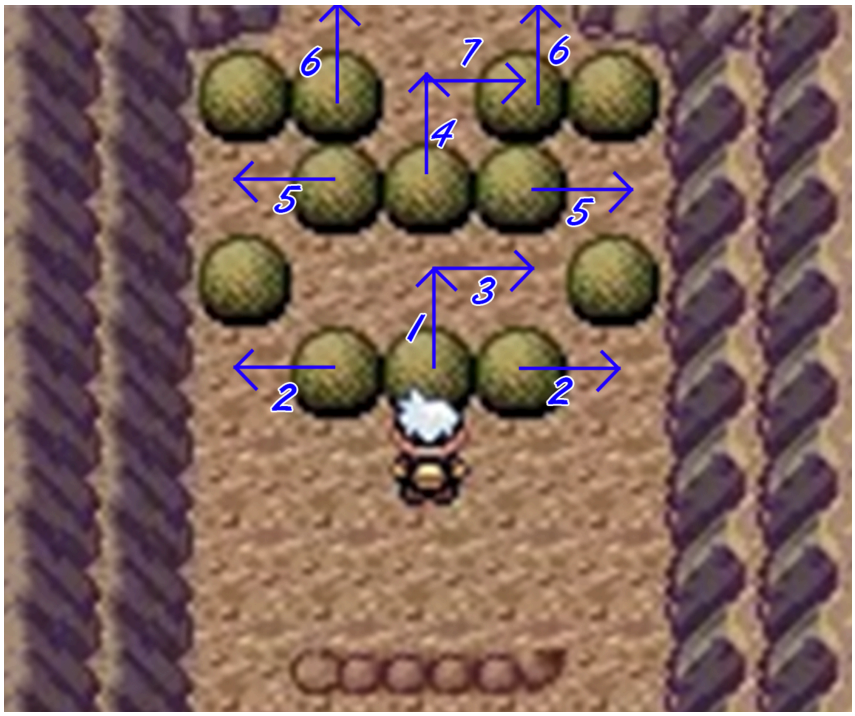

Figure 146: Image of a Pokémon pushing block puzzle, with its solution annotated.

The boss battle in this world would take place at the heart of the volcano, similar to one of the boss battles against an enemy called Groudon in Pokémon Emerald.

Figure 147: Image of the Groudon fight in Pokémon Emerald.

6.4.3. Final Boss World

The final world would be designed to test everything the player had learned from the 4 major worlds through a final combat section against a cyborg John McNamara, utilizing Skills Build Questions from all 4 previously fought bosses!

Carrying on with the main theme from the Hub World, this area would utilize a space theme, where tiles from every world would be used to line a path leading to the final boss, showcasing how all 4 worlds have come together to make one final section.

6.4.4. Other World Ideas

There were a number of other concepts that weren't implemented into the game, however, these puzzles and ideas may find their way into a future update!

6.4.4.1. World Theme

Although a theme has not been decided, we wanted to implement a cyberpunk or synthwave world. Cyberpunk is a genre of science fiction that focuses on a combination of high technology and low quality of living, for example cyborgs and artificial intelligence, juxtaposed with societal decay [7]. Synthwave, on the other hand, is a genre of music, that often draws inspiration from graphics seen in the 1980's, such as neon, wireframes, video cassettes and more as part of a visual aesthetic [8].

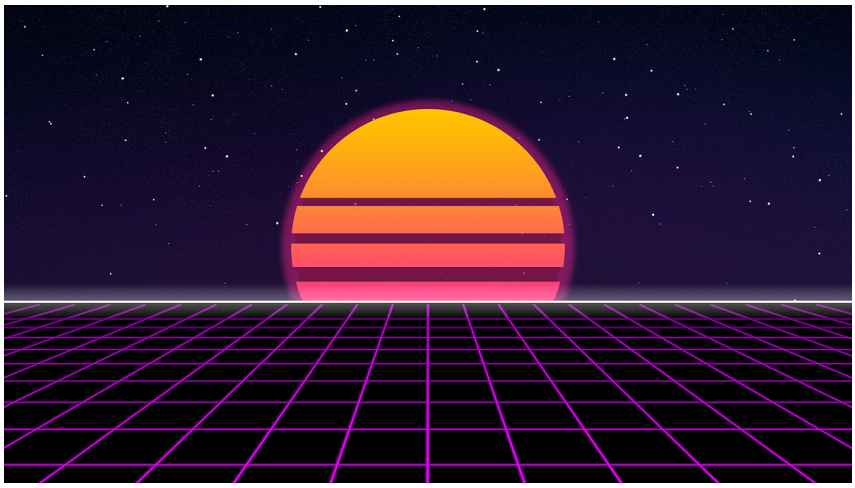

Figure 148: Image of the synthwave aesthetic.

6.4.4.2. Dance Dance Revolution Puzzle

Dance Dance Revolution is a series of arcade video machines, where the player is required to step on the specified locations on the floor of the machine. The floor of the game is partitioned into four sections using the four cardinal directions with symbols pointing in each direction. The game indicates the section of the floor on which the user is to step by presenting luminous arrows on the screen that point in the direction corresponding to the relevant section of the floor. Dance Dance Revolution presents the on-screen arrows as travelling from one side of the arcade machine's screen to another, entering a virtual box. When the arrow symbol enters this box, the user is to step on the corresponding section of the machine's floor. The game synchronises these events to music so the movement of the player's feet will roughly follow the beat of the song that is playing which mimicks dancing hence the name "Dance Dance Revolution".

We adapt Dance Dance Revolution to a computer game by using the arrow keys as digital analogues to the sections of the arcade machine's floor.

Figure 149: Two young men playing Dance Dance Revolution on an arcade machine.

6.4.4.3. Conveyor Belt Puzzle

A conveyor belt puzzle is a type of puzzle in which the player wants to get to the other side of the room, but to do so they need to step on a series of conveyor belts, when you step on one of these, they will send you to the other end of the conveyor belt, not allowing you to get off. You need to step on the correct conveyor belts to get to the other side of the map, and if you step on the wrong one, you get sent backwards instead.

This type of puzzle can be implemented without conveyor belts too, for example in Pokemon Emerald, you must ride on a series of water currents. In the image below, the green arrows show how taking a wrong path could lead to being sent back, and the red arrows show the correct path.

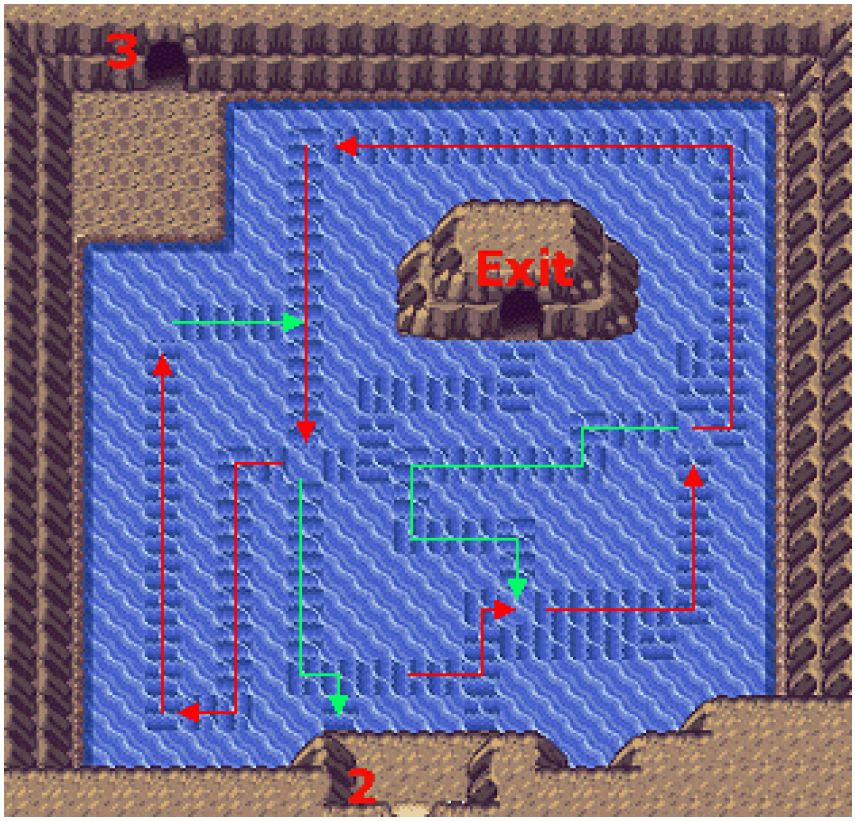

Figure 150: Image of a Pokemon Emerald's Seafloor Cavern water current puzzle.

6.5. Ethical and Societal Impacts

Our project, whilst only being a game, might have some potential ethical issues that need to be considered. On the contrary, the project might also have some potential upside when it comes to societal impacts.

6.5.1. Ethical Impacts

Games, by the nature of their existence, are meant to be a form of interactive entertainment that provide users with artificial challenges to overcome in order to reach a final goal [9]. This greatly mimics the concept of studying where, for example in subjects like mathematics, students are given problem sets to solve in order to pass an examination.

Combining the two might be an appealing concept at a first glance, since you get a system that can both teach useful content while also being fun and engaging, however we must be careful of the downsides of gaming. Gaming, when in long sessions and frequently, have been shown to lead to adverse health effect such as causing addiction and lead to increased aggravation. Games which have lots of flashing images also have known to cause epileptic seizures for photosensitive individuals with epilepsy [10].

Our game only has around one to two hours of play time at most, and has features such as saving and pausing, allowing for users to take a break whenever they wish. This alleviates the issue of long gaming sessions. The issue of frequent play is also taken care of, as the game offers little in the way of replayability (which is not a problem for an RPG as it should be a one-time experience) besides refreshing on the contents of Skills Build courses.

Another ethical issue that might arise is with respect to diversity. Any game should include in it a diverse set of characters, ideals, backgrounds, and perspectives, as this leads to a more engaging and enjoyable experience for all people regardless of their attributes [11]. However, implementing this in a game can be a daunting task, considering how much more research and expertise is necessary to appropriately represent diversity.

For our game, this mainly affects the choice of sprites for the main and side characters. Given that these are hand-crafted, a lot of work has to go into designing each character, and due to the time constraints we have not been able to fully implement other versions of the main character besides male, such as a female or a non-binary variant. Despite that, we have tried our hardest given the time constraints to make a set of diverse characters.

6.5.2. Societal Impacts

In its current iteration, the game serves as a tool to be used alongside Skills Build to learn the content in a way that is less formal and more enjoyable. Perhaps users will be more inclined to use the website and start a course due to it having a game associated with. Nevertheless, the scope of impact is limited to Skills Build users are interested in gaming, and who want to specifically learn the courses covered in the game.

By implementing our project on a wider scale (like having a section in-game for each badge) and integrating more closely the course content (beyond simply linking the Skills Build badge) whilst also being accommodating to users who have not had lots of experience gaming, we could elevate the scope of impact outside the previously mentioned restrictions, allowing almost all Skills Build users to benefit from the positive effects of gamification in learning.

6.5.3. Copyright Concerns

Our game uses various copyrighted assets such as music and some images. Since this game is for educational purposes, and we will not be charging people to play the game, we are mostly exempt from the copyright laws concerning the use of copyrighted assets.

If we were to monetize the game, however, changing the following list of assets would make our game copyright-free.

6.5.3.1. Audio

- 16 Bit SNES Super Street Fighter II Guile Stage Type B KDL3 Style AMK Commission
- Doki Doki Literature Club Just Monika
- Hollow Knight Queen's Gardens
- Last Bible 3 Underworld Forest
- Mario Luigi Bowser's Inside Story Mini Game 2
- Mario Sonic at the Olympic Winter Games DS Adventure Tours Blizland
- Mario Sonic at the Olympic Winter Games DS Dream Ice Hockey Fever Hockey
- OneShot Alula
- OneShot On Little Cat Feet
- Pokémon Diamond Pearl Platinum Route 216 Day
- Super Mario Galaxy 2 Soundtrack Sweet Mystery Yoshi
- Super Mario Galaxy 2 Soundtrack World 2
- Super Mario Galaxy 2 Soundtrack World 4
- Super Mario Galaxy 2 Soundtrack World 6
- South Park End Credits
- Doom Eternal - The Only Thing They Fear Is You
- Metroid Prime - Elevator Theme
- Metroid Prime - Item Room
- Metroid Prime - Samus Aran's Appearance Fanfare
- Metroid Prime - Tanon Overworld Depths
- Metroid Prime - Escape Form Frigate Orpheon
- Terraria Calamity Mod Music - "Guardian of The Former Seas" - Theme of Desert Scourge

6.5.3.2. Art

- Deltarune Spamton sprite
- Doki Doki Literature Club Monika
- Doki Doki Literature Club Dialogue Box
- Sprite Mancer Ice Cave

7. Glossary

In this section, we include some terms which are domain specific to game development. Having this separate section avoids the need of explaining a term in the middle of a sentence.

Player: The person controlling the game. Similar to how the person controlling a car is a driver.

Player Character: The character inside the game that the player controls.

NPC: Non-Player Character, any character in the game that is not the playable by the user.

Gamepad: A gamepad is a type of video game controller held in two hands, where the fingers (especially thumbs) are used to provide input [12].

Level: A level is any space available to the player during the course of completion of an objective.

World: A world is a level or collection of levels with a specific theme.

Portal: Inspired by the concept of a wormhole, it is a spacial distortion that will transport the player to a different world.

Sprite: A two-dimensional image or animation that is integrated into a larger scene or game environment that often represent characters, objects, or special effects within a game [13].

Sprite sheet: A single image file that contains multiple frames or animations of a sprite [13].

2D: A graphic rendering technique from a two-dimensional perspective. Usually referring to a top-down or side-ways view of a game [14].

HP: Hit Points or Health Points, which is a measure of how healthy your character is in a game [15].

Enemy: A non-player character that tries to harm the player [14].

Boss: An important enemy who is usually hard to defeat [16].

HUD: Heads-up display, which is the part of the screen that shows information such as the player's score or health [17].

Gamify: A strategic attempt to make an activity more like a game in order to make it more interesting or enjoyable [18].

Indie Game: Independent Video Games, created by "individuals or small teams who operate independently of major studios" [19].

Bibliography

[1] J. Pulham, J. Watson, C. Dubois-Veltman, S. Ezeh, N. Le, and A. Sirohia, “Group 17 Requirement Specification”. Nov. 23, 2023.

[2] C. Nash, “What is the very first RPG game?”. [Online]. Available: https://www.quora.com/What-was-the-very-first-RPG-video-game

[3] Wikipedia, “List of highest-grossing media franchises”. [Online]. Available: https://en.wikipedia.org/wiki/List_of_highest-grossing_media_franchises

[4] Nintendo, “Nintendo Pokémon Mystery Dungeon: Gates to Infinity (3DS) manual”. [Online]. Available: https://www.manuals.co.uk/nintendo/pok-mon-mystery-dungeon-gates-to-infinity-3ds/manual

[5] oloff3, “Playstation 2 Pad”. [Online]. Available: https://www.deviantart.com/oloff3/art/Playstation-2-Pad-83940856

[6] Unity, “Develop a modular, flexible codebase with the state programming pattern ”. [Online]. Available: https://unity.com/how-to/develop-modular-flexible-codebase-state-programming-pattern

[7] Wikipedia, “Cyberpunk”. [Online]. Available: https://en.wikipedia.org/wiki/Cyberpunk

[8] Aesthetics Wiki, “Synthwave”. [Online]. Available: https://aesthetics.fandom.com/wiki/Synthwave

[9] E. Adams, *Fundamentals of Game Design*. 2014. [Online]. Available: https://www.amazon.co.uk/Fundamentals-Game-Design-Ernest-Adams/dp/0321929675

[10] M. Griffiths, “Video games and health”. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC558687/

[11] Mohawk College, “The Importance of Equity, Diversity and Inclusion in the Video Game Industry ”. [Online]. Available: https://www.mohawkcollege.ca/about/news/blogs/importance-of-equity-diversity-and-inclusion-video-game-industry%C2%A0

[12] Wikipedia, “Gamepad”. [Online]. Available: https://en.wikipedia.org/wiki/Gamepad

[13] Lenovo, “What is a sprite?”. [Online]. Available: https://www.lenovo.com/us/en/glossary/sprite/#:~:text=A%20sprite%20is%20a%20term,special%20effects%20within%20a%20game.

[14] Wikipedia, “Glossary of video game terms”. [Online]. Available: https://en.wikipedia.org/wiki/Glossary_of_video_game_terms

[15] PLARIUM, “What Is the Meaning of HP and What Does It Stand For?”. [Online]. Available: https://plarium.com/en/glossary/hp/#:~:text=HP%20in%20games%20refers%20to,of%20the%20screen%20before%2C%20right%3F

[16] Cambridge Dictionary, “BOSS”. [Online]. Available: https://dictionary.cambridge.org/dictionary/english/boss

[17] Cambridge Dictionary, “HUD”. [Online]. Available: https://dictionary.cambridge.org/dictionary/english/hud

[18] Cambridge Dictionary, “GAMIFY”. [Online]. Available: https://dictionary.cambridge.org/dictionary/english/gamify

[19] N. Pajkovic, “What is an Indie Game? A Comprehensive Guide”. [Online]. Available: https://www.torontofilmschool.ca/blog/what-is-an-indie-game/#:~:text=Indie%20games%20stand%20for%20%E2%80%9Cindependent,studios%2C%20both%20financially%20and%20creatively.