Computer Science with Games Technology BSc

The Development of a 2D Platformer in Unity that Implements a Unique Invisibility Feature

By William Dibble

william.dibble@city.ac.uk

Consultant: Jacob Howe | Project Proposed By: William Dibble | Word Count: 11,996

Contents

Abstract

This project presents the development of an innovative 2D platformer game using the Unity game engine and C#, focusing on implementing unique mechanics that distinguish it from traditional platformers. The game offers a refreshing and engaging gameplay experience, featuring diverse levels, obstacles, enemies, and collectables while maintaining a user-friendly interface and robust character controls. The project incorporates Aron Granberg's A* Pathfinding Project to enable intelligent navigation for non-player characters in the game world, demonstrating its adaptability to a 2D platformer context. In addition to the development process, this report comprehensively evaluates the game's success in achieving its objectives and delivering a captivating experience for players. The evaluation is based on player testing, feedback, and analysis of the game's performance, highlighting areas for potential improvement and future iterations. The results demonstrate the impact of innovative mechanics on player engagement and enjoyment, offering valuable insights for game developers, researchers, and educators in game design and development.

1. Introduction

1.1. Problem Description

In a gaming industry flooded with 2D platformer games, the lack of originality in gameplay and features has led to market saturation and a decline in player interest. To rekindle excitement and offer a fresh experience, this project aims to develop a unique 2D platformer that introduces innovative features rarely seen in modern games. These unique mechanics will keep players engaged and eager to explore the game's stand-out aspects, making it a must-play for fans of the 2D genre.

The primary innovative feature is the ability for the player to turn invisible at will, adding an exciting element of stealth to the gameplay. This mechanic allows players to bypass some enemies undetected, offering new and strategic ways to progress through levels. However, impaired vision while invisible presents a thrilling challenge.

Additionally, a bullet collecting and switching system allows for diverse combat strategies. Drawing inspiration from successful platformers such as Super Meat Boy and Super Mario, the game aims to create a captivating experience with inventive features.

1.2. Objectives

The main objective of this project is to create a one-of-a-kind and enthralling 2D platformer game that stands out in a saturated market by implementing new features, such as the ability for the player to turn invisible, a bullet collecting and switching system and advanced AI for enemies.

Sub Objectives:

- Implement a game mechanic for the player to activate the invisibility feature and create a visual impairment effect to counteract the powerful nature of this feature.
- Implement a system for the player to pick up and use different types of bullets in combat. Doing so would include creating various bullets with diverse properties and a mechanic for the player to collect and switch between them.
- Create levels and enemies that are challenging but fair to the player. Doing so would include designing levels with obstacles and enemies that challenge the player but are not impossible to overcome.

- Create a story and visual aesthetic that is engaging and immersive, that includes developing a narrative for the game and creating artwork and sound effects that are visually pleasing and enhance the player's experience.
- Create enemies with advanced and dynamic AI, which changes their behaviour depending on whether the player is invisible or not.
- Test the game with a group of players to gather feedback and make improvements. Doing so will include arranging playtesting sessions with players with varying experience in video games and gathering feedback on the gameplay, level design, art, and sound. Using this feedback, I can make improvements to the game before release. I will need to research different playtesting methods and gather feedback in game development.

1.3. Design

To incorporate the iterative design process in my final project, I will break down the game development sub-objectives above into three stages. This systematic approach will allow me to systematically improve the game at each stage, ensuring the final product is of the highest quality.

Build 1: Basic Functionality

In the initial phase, I will lay the groundwork for the game by developing its core mechanics, including the invisibility feature and visual impairment effects. This will involve exploring different techniques to incorporate invisibility and visual impairment within Unity. Moreover, I will create a game prototype as proof of concept for the essential mechanics. This includes designing a basic player character with invisibility capabilities and developing a rudimentary level for testing purposes. I will also establish the game's fundamental visual framework, incorporating user interface and menu systems.

Build 2: Main Functionality

During the second phase, I will further develop gameplay components such as the bullet collection and switching system, designing engaging and balanced levels, and crafting enemies with dynamic AI that adapts based on the player's invisibility status. Additionally, I will develop diverse bullet types with distinct properties, like bouncing or sticky bullets, to provide the player with various combat options. I will also create a series of levels that offer a fair challenge to the player and devise unique and captivating enemies.

Build 3: Extra Functionality

I will elevate the game in the final phase by integrating advanced features, such as new intricate obstacles, improved visuals, and music. This stage will incorporate user feedback to enhance gameplay, complexity, and player engagement. If I have time, I will develop a third level with a double jump ability, providing players with more manoeuvrability and a challenging gameplay experience. Furthermore, I will design a formidable boss fight, smooth level transitions, and immersive music to create a captivating and satisfying player experience. Ultimately, I will polish and fine-tune the game to ensure a high-quality final product.

1.4. Beneficiaries

The expected beneficiaries of this project include:

- Myself, as the developer of this project, as it will provide me with learning opportunities in game development and the industry.
- Players who are looking for new and unique 2D platformer games to play, as this game would provide them with a new gaming experience that they can enjoy.
- The 2D platformer game industry, as it would bring a new and unique game that adds to the variety of games available in the market.
- The game development community, as it will contribute to the knowledge base of game development with the implementation of unique features such as the invisibility ability mentioned.
- Potentially, it could also have educational benefits for young players by incorporating educational elements such as problem-solving and critical thinking. This game could be a fun and interactive way to teach problem-solving skills and critical thinking.
- Additionally, by potentially incorporating educational elements such as problem-solving and critical thinking, the game could have potential educational benefits for young players.

1.5. Assumptions and Limitations

Assumptions made for this project include:

- My personal experience with Unity and 2D platformer game development will be sufficient to overcome most of the potential difficulties faced during this project. This includes my ability to code in C#, use Unity's engine, and understand the basics of 2D platformer game development such as player movement, collision detection, and level design.

- The resources available online for Unity and C# game development will be sufficient to complete the project to a high standard. This includes tutorials, documentation, and forums that can provide guidance and solutions to any problems that may arise during development.
- There will be a sufficient number of players interested in playing a 2D platformer game with my unique features. This includes players who are fans of 2D platformer games and are interested in unique gameplay mechanics that haven't been seen in modern platformer games before.
- Access to playtesters is possible, as it's crucial to have a group of players test the game and provide feedback. This includes recruiting playtesters from online communities, friends, or family.

Limitations for this project include:

- The project scope may need to be limited due to time and resource constraints. This includes potentially cutting certain features or levels to meet a specific deadline.
- The invisibility feature may prove too challenging to implement or may not be well-received by players. This includes the potential for technical difficulties in coding the feature or player feedback indicating it is not enjoyable or doesn't add enough to the game to keep it interesting.
- The game development and testing process may require more time and resources than anticipated. This includes the potential for unexpected bugs or issues to arise during development or testing that can prolong the process.

2. Output Summary

2.1. Unity Project Files

File Type: Unity Project Folder (Size: 1.78 Gb), Executable Folder (Size: 115 Mb)

Overview: The Unity project contains all the necessary files to compile and create a 2D platformer game with innovative mechanics. Approximately 90% of the assets I used were developed by myself, either from scratch or by modifying royalty-free assets.

Intended Recipients: University, dissertation supervisor, markers, Unity community, developers, and gamers.

Usage and Benefits: The project evaluates the success of the objectives, offers a fresh gaming experience, and inspires developers.

Results and Appendices: These files are located in the package submission. Reused resources are stated in Appendix B.

2.2. Project Evaluation Report

File Type: .pdf

Overview: A comprehensive analysis of the 2D platformer game's success in meeting objectives and providing a unique and engaging experience.

Intended Recipients: University, game developers, researchers, or educators.

Usage and Benefits: The report assesses the project's success and offers insights for future research or development.

Results and Appendices: The current document is the evaluation report.

2.3. Test Cases (Contained in Project Evaluation Report)

File type: Included in Project Evaluation Report (.pdf)

Overview: Guided evaluation scripts are used to test different aspects of the game after each build and integrated into the project evaluation report.

Intended Recipients: Myself, the project supervisor, and markers.

Usage and Benefits: Ensures objectives are fulfilled and adds insights into evaluation when grading the project.

Results and Appendices: Test cases are located in Appendix C.

2.4. Player Feedback and Results (Contained in Project Evaluation Report)

File type: Included in Project Evaluation Report (.pdf)

Overview: The project evaluation report includes feedback from players on the gameplay experience, focusing on innovative mechanics and enjoyment.

Intended Recipients: Myself, the project supervisor, and markers.

Usage and Benefits: Evaluate game performance, identify improvement areas, and provide result evidence when grading the project.

Results and Appendices: Full summary of feedback is in Appendix D under a dedicated section.

2.5. Video Demonstration and Code Overview

File Type: .mp4

Overview: 13-minute video showcasing the game, highlighting key features and gameplay elements, and a brief discussion of a chosen implementation aspect.

Intended Recipients: University, dissertation supervisor, markers, game development community, researchers, and educators.

Usage and Benefits: Visually demonstrates the game and its mechanics, supports project evaluation, and demonstrates objective achievement.

Results and Appendices: The video is submitted as a separate file to this document.

3. Literature Review

3.1. Application Domain

Video games have become a dominant form of entertainment, leading to a thriving, multibillion-dollar industry. The 2D platformer genre has played a foundational role in video game history and has evolved significantly from its early days with games like Super Metroid to modern classics (Blow, 2004). The genre offers a distinctive gameplay style, engaging players with levels filled with obstacles and enemies, immersing them in the game's narrative and world.

The widespread popularity of the 2D platformer genre has resulted in market oversaturation. To distinguish their games, developers must introduce innovative features that set their titles apart (Kasurinen & Knutas, 2018). In this context, incorporating the ability to turn invisible and a bullet collecting and switching system are two inventive features that can elevate the 2D platformer genre.

3.2. Examples of Similar Video Games

New Super Mario Bros, Super Meat Boy, and Cuphead are three successful modern 2D platformer games, each showcasing its distinct style and gameplay mechanics (Thorsen, 2010; Yin-Poole, 2012; Wood, 2017). They are well known for being challenging games that take practice and skill to master. Analyzing these games offers insight into the current market state and how my game can differentiate itself.

New Super Mario Bros delivers classic 2D platformer gameplay focusing on exploration, precise controls, and vibrant visuals. My game will also employ precise controls and exploration-based gameplay, but with the unique addition of an invisibility feature that enables players to evade specific enemies at the expense of limited obstacle avoidance capabilities (Thorsen, 2010).

Super Meat Boy is renowned for its challenging gameplay and precise controls. My game will also emphasise challenging levels with precise controls and skill-based gameplay. The game's enemies will be diverse, each with unique attack patterns to challenge the player. Additionally, my game will incorporate the invisibility feature and a bullet-collecting and switching system, further increasing the gameplay's complexity (Yin-Poole, 2012).

Cuphead is a unique 2D platformer employing hand-drawn animation and a 1930s cartoon aesthetic. My game will also emphasise challenging levels and boss battles but with a distinct visual style, utilising 16-bit textures to expedite development. The game will incorporate advanced enemy AI to present diverse and challenging encounters for the player (Wood, 2017).

3.3. Existing Literature

Reviewing existing literature and academic journals on game design, mechanics, and the player experience is crucial for developing an effective 2D platformer game. Researchers have emphasised the importance of balance, variety, and feedback in creating engaging gameplay experiences. In his book, 'The Art of Game Design', Schell says, *"One simple way to balance elements for fairness is to make sure that whenever something in your game has an advantage over something else, yet another thing has an advantage over that!"*. I need to ensure that players can approach my game with different strategies and methods of reaching the end whilst finding the advantages and disadvantages of these varying methods (Schell, 2008). When developing the enemy AI, I should consider how the invisibility feature affects the behaviour of the enemies towards the player.

Visual aesthetics and audio design also play critical roles in game design, as they can augment player immersion and enjoyment. Schell has also identified the importance of consistency, clarity, and uniqueness in visual design. Dynamic and adaptive audio can heighten player immersion and foster a more active and engaging gameplay experience. Rogers mentions in his book 'The Guide to Great Video Game Design' that *"Games tend to come to life when sound is added to them."* (Rogers, 2014). Taking care when creating or selecting sound effects and music is crucial to immerse players in the game and maximise replayability.

By applying these studies' findings to the design and coding of a 2D platformer game, developers can create a game that is well-received by players and stands out in a competitive market.

In conclusion, this literature review investigated the 2D platformer game genre and the game design features contributing to a successful gameplay experience. By examining three successful modern 2D platformer games, I have identified unique features that can distinguish games within the genre. Specifically, the invisibility ability, sound design, and advanced enemy AI have been identified as crucial design features contributing to these games' success.

Moreover, academic literature has underscored the importance of effective game mechanics, level design, visual aesthetics, and audio in creating an engaging and immersive gameplay experience. By incorporating these findings into the design and coding of a 2D platformer game, developers can create a game that is well-received by players and stands out in a competitive market.

By examining the existing literature on 2D platformer games and game development, I have acquired valuable insights into the unique features that make the genre popular and how these features can be further developed to create new and exciting gaming experiences. By combining these insights with my game's unique features, such as invisibility, challenging enemies and obstacles, and a distinct visual style, I hope to create a game that offers players a fresh and engaging gameplay experience. By doing so, my game will have the potential to carve out its niche within the 2D platformer genre and provide players with a memorable and enjoyable gaming experience.

4. Method

4.1. Methodology Introduction

In this chapter, the methodology used in the development of the 2D platformer game for the BSc Computer Science dissertation is described. This section provides a detailed and objective report of the work undertaken, including the analysis, design, implementation, and evaluation activities. The project lifecycle and general approach describe the three build stages. The testing and evaluation methods, as well as the quality assurance plan, are also discussed.

4.2. Project Lifecycle and Approach

The project followed an iterative development approach, similar to the Agile development model. This approach was chosen because it allows for continuous improvement, flexible planning, and incorporating feedback from playtesting sessions. The project was divided into three main build stages, each with specific objectives and tasks.

4.3. Tools

Unity (Game Engine)

Unity is a popular free-to-use game engine used to build games and create real-time 3D renders.

Visual Studio Code & Visual Studio 2022 (Programming IDEs)

Unity is compatible with Visual Studio Code and Visual Studio 2022, making them the best C# editors for this project. I have used both these software's throughout my time at university, so I was comfortable taking on these tools.

Photoshop 2023 (Image Editor)

Photoshop 2023 is arguably one of the top image editing software on the market. I have years of academic and professional experience using it, so it was my go-to tool for creating the textures for my game.

Google Drive (Backup)

Google Drive is an industry-leading Cloud file storage solution. I used it to back up all my project files to ensure that no data was lost due to corruption and that the files could be accessed from anywhere with a reasonable internet connection.

Adobe Audition (Sound Editing)

For quick and easy sound edits, I used Adobe Audition, as the workflow can be faster when working on smaller sound design tasks.

Google Forms (Evaluation and Feedback)

Google Forms is a tool built into the Google Software Suite; it is a free solution allowing simple surveys to collect unlimited responses. This will be used to write the surveys quickly.

4.5. Build 1 (Foundation Development)

In Build 1, the main objective is to create a basic version of the game, which lays the groundwork for core mechanics and gameplay. This stage emphasises setting up key components and features to form a prototype for testing and iterative improvements in future builds. I will assess each development decision to ensure effective implementation.

4.5.1. Develop Terrain with Collision Detection (Objective 1.1)

The first step involves constructing a basic terrain layout with collision detection, enabling interaction between the player, game objects, and the environment (Coding in Flow, 2021). The terrain will be connected to a ground layer for the subsequent player and enemy collision interactions. Effective collision detection is vital for a polished gameplay experience, preventing frustrating glitches or errors. As Khalifa explains in 'Level Design Patterns in 2D Games', Super Meat Boy uses Hallways to guide players on where to go, and in Super Mario World, stacked platforms drive the player upwards (Khalifa, 2019). Level design must be carefully considered.

4.5.2. Design a Player with Movement Capabilities and an Invisibility Trigger (Objective 1.2)

I will design a player character with fundamental movement mechanics like walking, running, and jumping (Coding in Flow, 2021). Additionally, I will implement an invisibility mode that enables the player to become invisible and visible again through a dissolve effect. This invisibility feature will add an extra layer of strategy and challenge to the game, allowing players to avoid detection and overcome certain obstacles in a unique way.

The game will activate different animation states based on player actions, with the terrain's ground layer acting as the player's floor for jump detection. Following the advice of Rogers, *"When you are designing controls, try establishing design rules for your control schemes based on hand placement"* (Rogers, 2014). With this in mind, I will be sticking with using the arrow keys to move left and right, the spacebar to jump, and because F, C and V are all next to each other, these will be used for turning invisible, shooting the gun, and switching bullets, respectively.

A specific key press will activate the invisibility mode and will involve changing the player's material properties to create a smooth dissolve effect. Audio cues will also be crucial in indicating the transition between the visible and invisible states (Collins, 2013).

4.5.3. Implement a Camera that Follows the Player and Creates a Parallax for the Background (Objective 1.3)

A camera system will be established to follow the player's character, keeping them at the screen's centre for a seamless and engaging gameplay experience. A parallax effect will be created for the background, adding depth and movement to the game world. Developing a well-implemented camera system and parallax background is essential for player satisfaction and overall game quality.

4.5.4. Integrate Basic Sound Effects for Player Actions (Objective 1.4)

I will incorporate basic sound effects corresponding to player actions to improve game immersion. These effects will be activated when necessary. Integrating basic sound effects for player actions enhances the game's immersion will add depth to the overall experience. Sound effects will help reinforce the player's actions and provide feedback on their interactions with the game world and the bullet collection, leading to a more satisfying gameplay experience (Collins, 2013).

4.5.5. Create Dangerous Obstacles for The Player to Avoid (Objective 1.5)

I will design various obstacles to present challenges to the player and enhance gameplay depth, with proper functionality ensured through dedicated scripts. The obstacles will allow players to test their skills, develop strategies using their invisibility feature, and overcome challenges (Coding in Flow, 2021). Well-designed obstacles will contribute to the player's sense of accomplishment and progression throughout the game.

4.5.6. Create Enemies with Simple AI (Objective 1.6)

Simple enemy characters will be introduced with AI behaviours, offering an additional challenge. Developing enemies with basic AI will be vital in introducing variety to the gameplay experience. Simple enemy AI behaviours, such as pathfinding movement towards the player, can create dynamic and engaging encounters, encouraging players to adapt their strategies and keeping the experience fresh and exciting.

I have decided to use Aron Granberg's A* Pathfinding Project to achieve effective AI pathfinding for the enemies. The A* Pathfinding Project is a widely used and powerful tool for implementing pathfinding in Unity-based games. It offers efficiency, flexibility, customizability, and a strong community with plenty of documentation. The free version does more than enough for what I will need it for (Granberg, 2021).

4.5.7. Create a Respawn System for The Player (Objective 1.7)

I will set up a respawn system to maintain game flow and reduce frustration. This system will enable the player to restart from the beginning of the level they are on when they die. Establishing a player respawn system is crucial for maintaining game flow and reducing player frustration. It promotes continued gameplay preventing a player from becoming discouraged by setbacks that take them back to the beginning of the game.

4.5.8. Implement a Level Start and a Finish (Objective 1.8)

Distinct starting and ending points will be defined for the level, providing a sense of progression and achievement. Clearly defined starting and ending points create a structured gameplay experience and contribute to a sense of accomplishment when players complete a level.

4.5.9. Implement a Bullet Collecting and Switching System (Objective 1.9)

I will establish a mechanism for collecting and switching between various bullet types, allowing players to adapt their strategy for different situations and enemy encounters. Doing so will introduce strategic depth and variety to the gameplay, adding a layer of challenge and complexity. The player must cater their play style to work with the different bullet features available.

4.6. Build 2 (Core Functionality)

Build 2 focuses on refining the game's core functionality, including bullet collection and switching, advanced enemy AI, level design, and other critical components.

4.6.1. Create a Tutorial for The Player (Objective 2.1)

A tutorial level will be developed, introducing players to essential mechanics and features. This is essential in helping ensure that players can effectively engage with the game's features and controls, leading to a more enjoyable and satisfying gameplay experience. As Schell mentions, *"Many game tutorials are terrible — ones created by this method are likely to have an aura of excellence about them"* (Schell, 2008). By saying "this method, " he refers to a system in which many playtesters are used to identify how practical the tutorials are. He slowly makes modifications as feedback is gained until it becomes perfect, which I will certainly consider in my development.

4.6.2. Create a Start Menu for The Player to Start the Game (Objective 2.2)

I will develop a start menu, allowing players to start the game, access the tutorial, and view credits. Creating a start menu interface is crucial for providing players with a polished and professional game experience. A well-designed start menu sets the game's tone, such as the pixel art style, it allows players with little experience to start with the tutorial, or experienced players can jump right into the first level and learn the controls and mechanics as they go.

4.6.3. Create a Second Level with a Boss Fight Element (Objective 2.3)

In his book "What Video Games Have to Teach Us About Learning and Literacy", James Gee explains, *"The game repeatedly confronts players with a similar type of problem… until players achieve a routinized, taken-for-granted mastery of certain skills. Then the game confronts players with a new problem, for instance, a new type of enemy or a boss, which forces the players to rethink their now taken-for-granted mastery"* (Gee, 2003). This has influenced me to create a second level featuring a boss fight to test players' platforming and invisibility skills when high pressure is applied, as the Build 1 enemies wouldn't do this. Creating a second level with a boss fight introduces a significant challenge for players to overcome, testing their platforming and invisibility use skills.

4.6.4. Improve the AI for The Enemies (Objective 2.4)

When discussing enemy AI behaviours, Rogers states, *"The best solution doesn't include unpredictability. For Crash Bandicoot 2, Naughty Dog tried creating more behavioral AI with fewer simple patterns. The players liked the challenge of figuring out the enemy's patterns"* (Rogers, 2014). I will improve the game's enemy AI by incorporating dynamic behaviour based on the player's visibility status, including patrolling, searching, and reacting differently to the visible or invisible player. This way, the player can learn how to manipulate the enemy's behaviour to benefit them by analysing their patterns and adjusting their strategy. Enhancing enemy AI should improve the game's overall challenge and engagement, as dynamic enemy behaviour based on the player's visibility status creates more diverse and exciting gameplay scenarios. It adds use for the invisibility feature, leading to a more enjoyable and replayable game experience. Players must adapt their strategies and tactics to effectively deal with the improved enemy AI.

4.6.5. Add Sound Effects for the Enemies and Environment (Objective 2.5)

Sound effects for various game elements will be added, such as enemy movement, attacks, and death animations. Additionally, environmental sounds, such as ambient background noise, will be incorporated to create a more immersive experience for the player. Adding sound effects for enemies and the environment enhances the game's atmosphere and overall immersion, contributing to a more engaging and enjoyable experience for the player. Well-designed sound effects can also provide important gameplay feedback, allowing players to react to events occurring within the game world more effectively. Furthermore, ambient background noise helps create a sense of realism and depth, making the game world feel alive and dynamic.

4.6.6. Implement a Timer Score System (Objective 2.6)

When discussing Lens #36: The Lens of Competition, Schell mentions, *"Determining who is most skilled at something is a basic human urge. Games of competition can satisfy that urge"* (Schell, 2008). To drive this urge, I will implement a timer score system to track the player's progress through each level, noting the overall completion time. This system will encourage players to complete the levels more quickly and efficiently, adding challenge to the gameplay. Implementing a timer score system serves multiple purposes. Firstly, it encourages players to improve their skills and strive for faster completion times, adding replayability and a layer of challenge to the game. Secondly, it allows for friendly competition among players, as they can compare their completion times with friends or on leader boards. Lastly, from a game design perspective, the timer score system can help developers identify potential bottlenecks or areas where players struggle, providing valuable information for refining and optimising level design.

4.6.7. Implement a UI for the Bullet System (Objective 2.7)

I will develop a user interface for the bullet system, displaying the player's current bullet type, remaining ammunition, and available bullet types. This UI will provide players with crucial information about their combat resources and allow them to switch between bullet types easily. Implementing a UI for the bullet system is essential for providing players with clear, easily accessible information about their combat resources. Doing so allows them to make informed decisions during gameplay, facilitating strategic choices and enhancing overall player agency.

5.3. Build 3 (Extra Functionality)

Build 3 will incorporate additional features into the game, including new abilities, more complex obstacles and adversaries, enhanced visuals, music, and overall refinement.

5.3.1. Implement a Third Enemy into Level 1 (Objective 3.1)

Based on the feedback received from user testing, I will design and implement a third enemy type for level 1. The enemy will have a unique behaviour pattern, differentiating it from the existing enemies in the level. This addition aims to enhance the gameplay experience, increase the level's complexity, and promote player engagement and enjoyment.

5.3.2. Create a Third Level with a Double Jump Ability and More Challenging Obstacles (Objective 3.1)

I will craft and incorporate a third level featuring a new double jump ability for the player, enabling them to access elevated platforms and evade more treacherous obstacles. This addition will grant players increased manoeuvrability and broaden the game's platforming mechanics.

5.3.2. Introduce More Complex Obstacles for the Third Level (Objective 3.2)

The third level will present more complex challenges, such as unseen blades and hidden spikes. These hazards will assess the player's double jump proficiency and overall platforming skills, offering a more demanding gameplay experience.

5.3.3. Create a Fourth and Final Level that has a Boss Fight and a Final Score Screen (Objective 3.3)

For level 4, I will devise a more formidable boss enemy making the ultimate boss confrontation. This adversary will be able to turn invisible, fly, and exhibit more advanced combat tactics than the boss from level 2.

5.3.4. Implement a Visual Aesthetic When Changing Levels (Objective 3.4)

In his book on game development, Rogers highlights, *"More recent games strive for a "seamless" loading-screen-free experience by disguising the loading screen with slowly opening doors, long elevator rides, or dissipating fog"* (Rogers, 2014). And whilst these ideas may take too long to implement, I can still create a visually appealing transition between levels to enhance the overall game experience. This transition could involve a combination of animations, screen effects, or audio cues, creating a smooth and engaging shift between levels. Implementing an aesthetic transition ensures a seamless flow from one level to another, maintaining player immersion and preventing abrupt interruptions in gameplay.

5.3.5. Incorporate Music into the Game (Objective 3.5)

Original background music will be composed and integrated into each level, as well as the menu and victory screens. The music will establish a unique ambience for each stage, set the tone, and enrich the overall player experience.

5.3.6. Polish and Fine Tune the Game (Objective 3.6)

I will dedicate my final efforts to refining and optimising the game, rectifying any outstanding issues, balancing gameplay components, and ensuring that all visual and audio assets are streamlined and consistent. This process guarantees the final product is high quality and delivers a gratifying and engaging player experience.

4.8. Testing and Evaluation

To ensure the game's success and to gather feedback on its functionality and design, the following testing and evaluation methods were employed:

4.8.1 Playtesting Sessions

As Schell states, *"A post-game interview is a great way to ask players questions too complex for a simple survey sheet. It's also a way to get a sense of how they really felt about the game, since you can see emotion in their faces and hear it in their voices"*. After I have completed development on Build 2, playtesting sessions will be conducted with diverse participants, primarily friends and family, to collect valuable feedback on gameplay, mechanics, and overall experience. Although using friends and family as playtesters may increase bias, it is a pragmatic solution given the limited resources available for conducting larger-scale, unbiased playtesting. Playtesting is an essential part of the game development process, as it helps identify potential issues and areas for improvement.

4.8.2 Qualitative Questionnaires

I will use Google Forms to create qualitative questionnaires to gather detailed feedback from playtesters. The questionnaires will focus on game mechanics, level design, user interface, and overall game enjoyment. Qualitative questionnaires provide valuable insights into player preferences and experiences, allowing for a deeper understanding of the game's strengths and weaknesses. Collecting player feedback through questionnaires enables the identification of issues that may have been overlooked during the development process, ultimately leading to a better final product.

4.8.3 Semi-Structured Interviews

I will conduct short semi-structured interviews with playtesters during the testing sessions. The questions will be adapted based on the participants' answers to gather more in-depth insights into their experiences with the game. Semi-structured interviews offer flexibility in the discussion, allowing for a more comprehensive exploration of the playtesters' thoughts and feelings. I will transcribe these interviews for further analysis. Conducting semi-structured interviews can provide additional insights into the game's design and usability, complementing the feedback obtained from qualitative questionnaires.

4.8.5. Data Analysis and Interpretation

The data gathered from the questionnaires and interviews will be analysed to identify common themes, trends, and areas for improvement. I will use this feedback to inform further development and refinement of the game, ensuring it is functional and enjoyable for players.

5. Results

5.1. Build 1 (Basic Functionality)

In Build 1, the primary goal was to develop a basic version of the game, which would serve as a solid foundation for the core mechanics and gameplay. I centred this initial stage on establishing essential elements and functionalities to create a prototype that I could test and iteratively improve upon in subsequent builds.

5.1.1. Create a Terrain with Collision Detection (Objective 1.1)

The design process began with me crafting a basic terrain layout incorporating collision detection, enabling the player and other game objects to interact with the environment as intended. I created a prefab for the terrain, to which a Tilemap Collider, Composite Collider, and Platform Effector were attached. I drew the terrain using the Tile Palette, and the collider automatically generated around it, streamlining the design process. I finally linked this terrain to the "Ground" layer for the later player and enemy collision interactions.

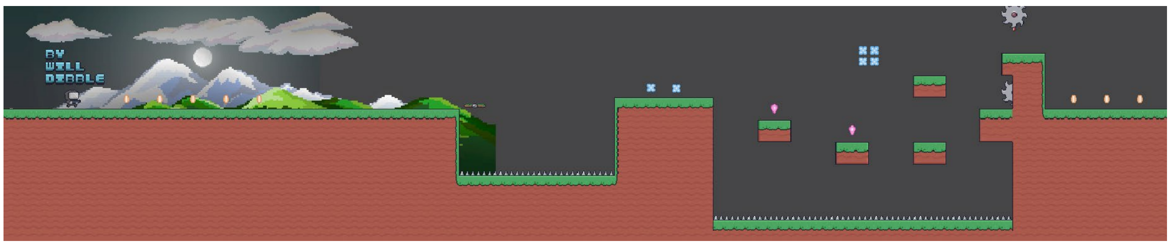


Image 1: Level 1

5.1.2. Implement a Player with Movement and an Invisibility Trigger (Objective 1.2)

I developed a controllable player character with movement mechanics like walking, running, and jumping. A new object was created for the player, incorporating a Sprite Renderer, Rigidbody, and Box Collider. A script for player movement captures "Horizontal" movement (arrow keys) and "Jump" (spacebar). Animation states trigger based on player actions. The terrain's "Ground" layer was assigned as the player's floor.

I implemented an invisibility feature with a dissolve effect using a custom material designed in Unity's material editor. The material has a "_Fade" property controlling the player sprite's transparency. A new script, "Dissolve.cs" (Appendix F, 1.29), manages the invisibility feature, defining a Material variable and a bool "isDissolving" to track the player's invisibility state. The Dissolve material can be found in Appendix G.

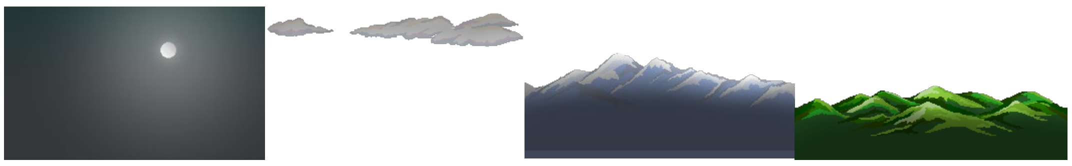
The PlayerMovement script obtains and initialises essential components such as Rigidbody2D, BoxCollider2D, and Animator. The Update() method processes player input for horizontal movement and jumping and updates animations. The UpdateAnimations() method evaluates the player's state and updates the Animator component. The player's rotation is adjusted to face the correct direction.

Image 2: Player Animated

The onGround() method uses a BoxCast to verify ground collision, which is essential for determining if the player can jump. The PlayerMovement script can be viewed in Appendix F (1.19).

5.1.3. Implement a Camera that Follows the Player and Creates a Parallax for the Background (Objective 1.3)

To create a dynamic and immersive gameplay experience, I have implemented a camera system that smoothly follows the player's character, keeping them at the centre of the screen throughout the game. The primary camera was assigned a Cinemachine camera, a built-in Unity feature that allows for customisation and fine-tuning of camera characteristics. Doing so involves setting up a bounding box that restricts the camera's movement, ensuring that untextured parts of the level remain hidden. I assigned a "Camera Tracker" child object to the camera that I could set the Cinemachine camera to follow, moving with the player's movement throughout the game.



Images 3-6: Parallax Background Layers

Using guidance from PavCreations (Pav, 2020), I implemented the parallax effect for the camera and background layers using three scripts: ParallaxCamera.cs, ParallaxLayer.cs, and ParallaxBackground.cs. These scripts work together to create a depth illusion by making background layers move at different speeds, giving the impression of a 3D environment.

The ParallaxCamera.cs script is attached to the primary camera, detecting the camera's movement and passing the change in position (delta) as a parameter. The event informs the other parallax scripts about the camera's movement so that they can adjust their behaviour accordingly. The script initialises the old position with the camera's position and checks if the camera's position has changed. If so, it triggers the onCameraTranslate event.

The ParallaxLayer.cs script is attached to individual parallax background layers. It listens to the onCameraTranslate event from the ParallaxCamera script and moves the layer accordingly based on the camera's movement and the parallax factor. The parallax factor controls the speed of layer movement and creates a depth effect. It calculates the new position of the layer based on the camera movement and parallax factor and updates the layer's position.

The ParallaxBackground.cs script is attached to a parent GameObject holding the parallax background images. It initialises and manages the parallax layers and its child GameObjects with ParallaxLayer components. The script listens to the onCameraTranslate event from the ParallaxCamera script, updating the positions of the parallax layers according to the camera's movement creating the overall parallax effect. These parallax scripts can be viewed in Appendix F (1.14, 1.15, 1.16).

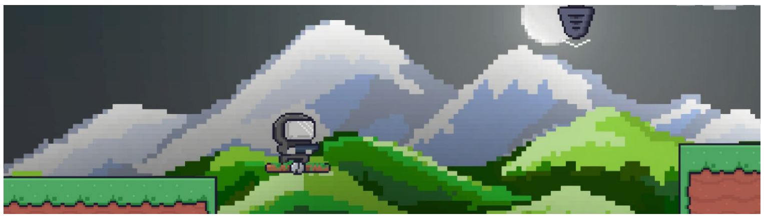


Image 7: Player In Front of Parallax Background

In summary, the camera system effectively follows the player's character and creates a parallax effect for the background, resulting in a smooth and responsive gameplay experience.

5.1.4. Integrate Basic Sound Effects for Player Actions (Objective 1.4)

To enrich the gaming experience and enhance the game's immersion, I implemented simple sound effects corresponding to the player's actions, such as jumping, death, and item collection. Mp3 and Wav files were assigned to new "Audio Source" components in an audio manager object and connected to relevant scripts, ensuring the sound effects would trigger when necessary. I modified royalty-free sound effects in Adobe Audition from Dagurasu’s

Retro Sounds (Dagurasu, 2021) and JDWasabi's Sound Effects Pack (JDWasabi, 2021) and utilised them for the audio manager.

The AudioManager.cs script was created and attached to a new GameObject for audio management. This script manages the game's audio, including audio source and audio clip variables for sound effects and background music, as well as methods for playing each sound effect and music for the current scene. The AudioManager script can be viewed in Appendix F (1.1). In the PlayerMovement.cs script, the AudioManager component is obtained by searching for an AudioManager instance in the scene using FindObjectOfType.

The AudioManager instance is then used to play the appropriate sound effects when certain events occur, such as the player jumping. The audio manager handles playing the sound effect through its PlaySound method, which uses AudioClip as a parameter.

Sound effects add depth to the game and provide auditory feedback to the player, resulting in a more engaging and immersive experience. AudioManager streamlines the audio management process and easily incorporates sound effects into the 2D platformer game developed in Unity.

5.1.5. Create Dangerous Obstacles for The Player to Avoid (Objective 1.5)

I designed and implemented various obstacles to create a more engaging and challenging 2D platformer game, adding depth to the gameplay. The obstacles included in the game are short spikes on the ground, spinning sawblades moving along a predefined path, and breakable platforms that collapse after the player touches them. Upon contact with these obstacles, the player character is killed. I developed dedicated scripts for each obstacle type to ensure proper functionality and seamless integration with the game.

The spinning sawblades are controlled by two scripts, PointFollower.cs and Rotate.cs, both attached to the sawblade prefab. PointFollower.cs manages the sawblades' movement between a series of points and can also be used for other game elements, such as moving platforms and the invisible bats that are implemented in build 3. The PointFollower script can be viewed in Appendix F (1.21). The Rotate.cs script handles the continuous rotation of the sawblades, only activating the rotation when the player is within a specified distance to optimise game performance.

Spikes are static obstacles positioned on the ground, with no requirement for movement or rotation. The PlayerLife.cs script, attached to the player object, detects collisions with these obstacles and manages the player's death accordingly.

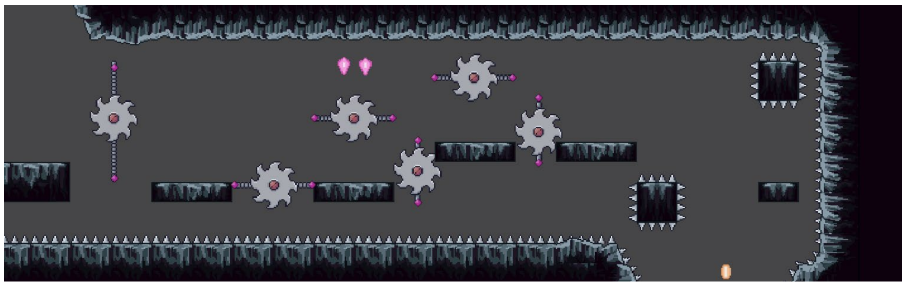


Image 8: Level 1 Cave Obstacles

Another obstacle implemented is the breakable platform, which appears as a regular platform but collapses after the player touches it, increasing difficulty and haste within the gameplay. This feature is controlled by the BreakablePlatform.cs script attached to the platform object. The BreakablePlatform script can be viewed in Appendix F (1.3). When the player collides with the breakable platform, a coroutine is initiated with a specified delay before the platform breaks. Upon breaking, the script instantiates a breaking platform animation object and plays a falling impact sound effect before destroying the breakable platform object.

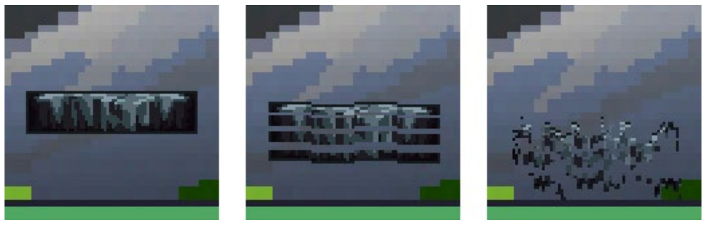


Image 9: Breakable Platforms

The breaking platform animation object is controlled by the FallingCavePlatformDestroy.cs script, which is responsible for destroying the animation object after it has played out. The script includes a delay variable to determine the time before the object is destroyed. The FallingCavePlatformDestroy script can be viewed in Appendix F (1.8).

5.1.6. Create Enemies with Simple AI (Objective 1.6)

Using Aron Granberg's A* Pathfinding Project, I implemented AI behaviour to develop simple enemy characters with pathfinding abilities. This approach enabled the calculation of the shortest path to the player, thus guiding enemy movement effectively, and enemies shared the same tag as obstacles, resulting in player death upon contact. I developed two distinct enemy types. The first was a jumping enemy, which follows parabolic jump paths determined by the pathfinding algorithm and has maximum jump power limits to prevent excessively large jumps. The second was a rolling enemy that slides along the ground towards the player at great speed.

The seeker script attached to the enemies is responsible for pathfinding and navigation, allowing the enemies to find and traverse the shortest path to the player. It is built using the A* Pathfinding Project (Granberg, A. 2021), a widely used AI navigation package in Unity.

The Seeker class is the main component of this script, handling the logic for requesting, processing, and drawing paths. It is designed to be attached to a GameObject in the Unity scene, usually an enemy character. The class contains multiple public and private fields, properties, and methods to manage pathfinding functionality.

The jumping enemy is controlled by the JumpingEnemyAI.cs script, which employs pathfinding to calculate and update the enemy's path towards the target while also utilising a jump trajectory instead of a path. This is calculated when the enemy is on the ground and within a certain distance from the target. Various components such as Rigidbody2D, Seeker, BoxCollider2D, and SpriteRenderer are used to manage the enemy's movement, pathfinding, collision, and appearance. The JumpingEnemyAI script can be viewed in Appendix F (1.11).

Image 10: Jumping Enemy

For the rolling enemy AI prefab, another script titled "RollingEnemyAI.cs" was developed. This script focuses on controlling the AI behaviour, which also follows a path to move towards the player's position using the Pathfinding package. The enemy's speed, the distance to the next waypoint, and the enemy's health are public variables. The script also includes functions for handling enemy health, death, and collisions with other objects. The RollingEnemyAI script can be viewed in Appendix F (1.22).

Image 11: Rolling Enemy

5.1.7. Create a Respawn System for The Player (Objective 1.7)

I developed a respawn system to maintain game flow and minimise frustration, primarily implemented through the PlayerLife.cs script, handling the player's life, detecting collisions with traps, managing death, and restarting the level. The LevelRestart() function is called after the death animation with a one-second delay.

The PlayerLife.cs script (Appendix F, 1.18) declares necessary components, references Timer and AudioManager scripts, and sets bullet counts based on the current level in the Start() method. The OnCollisionEnter2D() method detects collisions between the player and traps, calling the Die() method if not invincible. Die() sets the player's Rigidbody to static, triggers the death animation, disables the player's BoxCollider, and restores visibility if needed.

Lastly, the LevelRestart() method restarts the level by reloading the current scene and re-enabling the player's BoxCollider for interaction upon respawning.

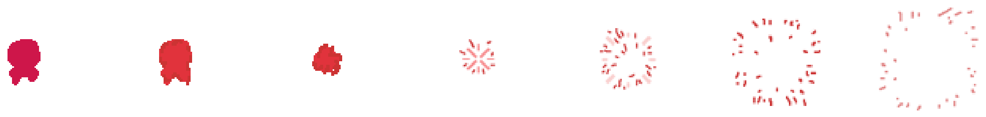


Image 12: Player Death Animation

5.1.8. Implement a Level Start and a Finish (Objective 1.8)

I implemented distinct level start and finish points in the 2D platformer game to give players a sense of progression and accomplishment. This structure helps players linearly navigate through the game world and reach the end. The first level featured various enemy types, spikes, sawblades, and moving platforms, presenting challenges for the player to overcome. The level design and elements were crafted to guide the player from the starting point on the left side of the map to the finish point on the right side.

The level is comprised of two primary sections with different environmental features and challenges. The first section is an outdoor grassy terrain with separated platforms, requiring players to jump and navigate through platforming obstacles such as the moving platforms.

This section introduces the player to hazards like spikes, sawblades, and moving platforms. It also exposes them to different enemy types, teaching the player early on about the game's combat mechanics and how to deal with hazards.

To facilitate level progression, scripts and game objects are utilised to manage level transitions, detecting when the player has reached the finish point and moving the player to the next level. The NextLevelTrigger script can be viewed in Appendix F (1.13). Trigger colliders are used to identify when the player has reached the finish point, while the LevelManager script manages level transitions and overall game progression. The LevelManager script can be viewed in Appendix F (1.12). Implementing clear level start and finish points gives players a sense of advancement and accomplishment, enhancing their experience. The two main sections within the first level, the outdoor and cave areas, showcase various challenges and environmental features that test the player's skill and understanding of game mechanics. The level design and scripting work together to guide players from start to finish, offering an engaging and rewarding gameplay experience.

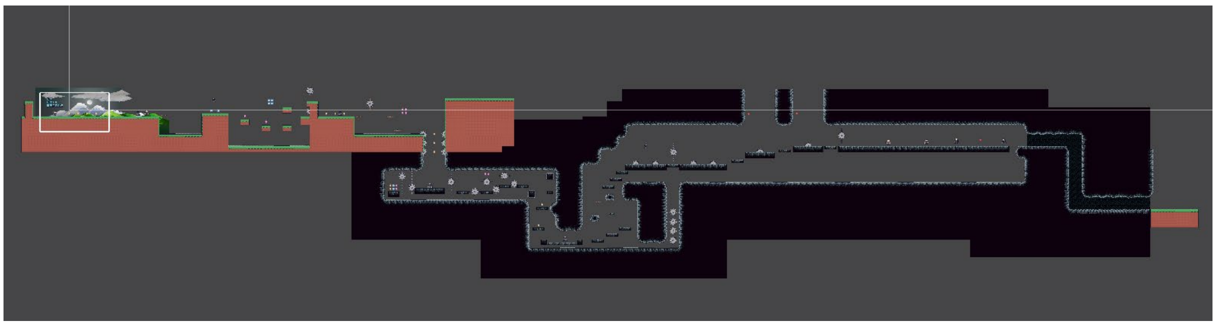


Image 13: Level 1 Complete

5.1.9. Implement a Bullet Collecting and Switching System (Objective 1.9)

I designed and implemented a bullet collecting and switching system to offer strategic choices during gameplay in the 2D platformer game developed in Unity. This system allowed the player to adapt to different situations and enemy encounters by providing them with an opportunity to collect and switch between various bullet types. The bullet pickups were designed as animated prefabs, which triggered a collision event when the player touched them, and the bullet counts on the user interface were updated accordingly.

I developed three bullet types to offer varying levels of utility and challenge, encouraging players to strategise and adapt their playstyle throughout the level. The bullet types are as follows:

- Orange bullet: This bullet maintains a consistent speed and trajectory as it travels through the air until it hits an object. The orange bullet has a medium speed.
- Blue bullet: Like the orange bullet, the blue bullet travels consistently through the air but burns up quickly. It deals more damage compared to the orange bullet.
- Pink bullet: The pink bullet moves through the air with a drop rate, causing it to fall to the ground and burn up after a short distance. This bullet deals the most damage among the three bullet types.

I implemented this system through two scripts: WeaponSelection.cs and Timer.cs. The weapon script handles selection and bullet counts and includes methods for spawning bullets, updating UI elements, and collecting bullets. On the other hand, the Timer.cs script defines a Timer class with functionality for managing bullet counts. It also has getters and setters for bullet counts and bullet level counts that keep track of the bullet counts from scene to scene. The WeaponSelection and Timer scripts can be viewed in Appendix F (1.26).

Image 14: Bullet Pickups

In the WeaponSelection.cs script, I created an array of weapon sprites and variables for UI elements, bullet prefabs, and references to the Timer and AudioManager scripts. The Start() method initialises the weapon index, bullet box image, and finds the Timer and AudioManager objects. The Update() method is responsible for updating the bullet count text for each type of bullet and handling input for switching weapons and firing bullets. When the player triggers a collision with a bullet collectable, the script checks the tag of the collided object and increases the corresponding bullet count. The script also updates the UI elements for the bullet counts, changing the text colour to red when it reaches its maximum value of 9.

The Timer.cs script is responsible for timing gameplay and managing bullet counts. It handles the "scene loaded" event by starting the timer when the Level 1 scene is loaded and stopping it when the Final Score Screen scene is loaded.

5.2. Build 2 (Main Functionality)

Build 2 required me to develop the main functionality of the game, including the bullet collecting and switching system, advanced enemy AI, level design, and other essential components.

5.2.1. Create a Tutorial for The Player (Objective 2.1)

I designed and implemented a comprehensive tutorial level that introduces the player to the core mechanics and features of the platformer. This tutorial level is essential for players to familiarise themselves with the game's mechanics and controls before progressing to the game's main levels.

The tutorial begins by presenting the player with on-screen text describing the movement controls for moving left and right and the controls for executing a jump. Doing so clearly and concisely explains the basic controls required to navigate the game environment. It then introduces the shooting mechanics by displaying text instructing the player to press the 'C' key to shoot and the 'V' key to switch between different bullet types. This information is vital for the player to understand and master the game's combat system.

Following this, the tutorial level incorporates a jumping enemy the player must shoot and defeat. This encounter lets the player practice the shooting mechanics and become comfortable with the game's combat system in a low-pressure environment.

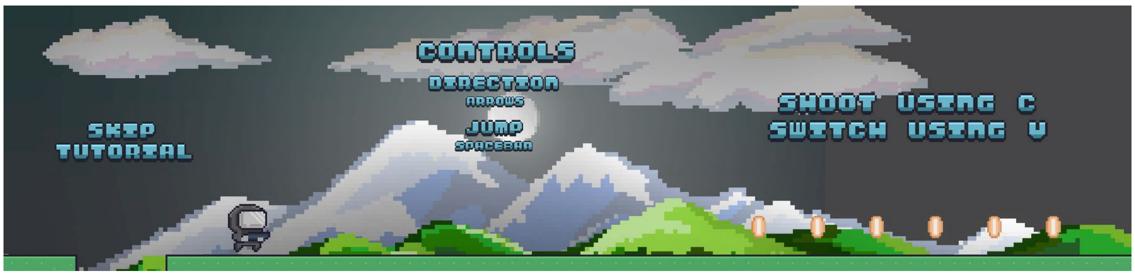


Image 15: Tutorial

In addition to the jumping enemy, a floating platform is placed above a pit of spikes. This element of the tutorial level challenges the player to utilise their newly acquired jumping skills to navigate the platform and successfully avoid the hazardous spikes below.

I have then introduced a small obstacle course consisting of moving saw blades and floor spikes to teach the player how consistently moving and strategising will keep them alive by refining their movement and jumping skills while avoiding deadly obstacles.

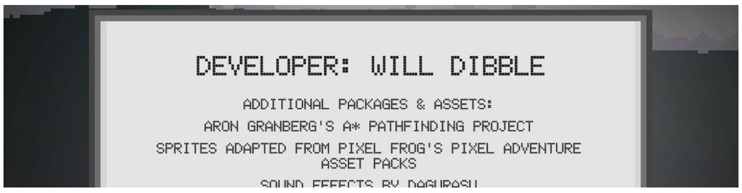
Lastly, the tutorial level concludes with a hole labelled 'Level 1', which, upon entering, transitions the player to the first main level of the game. Doing so clearly indicates to the player that they have successfully completed the tutorial and are now ready to embark on the game's main levels.

5.2.2. Create a Start Menu for The Player to Start the Game (Objective 2.2)

In the development process, I designed and implemented a start menu that allows players to interact with on-screen buttons to initiate the game, access the tutorial, and view credits. The start menu serves as the primary entry point for the game, providing a user-friendly interface for players to navigate through the game's various options.

One of the unique features implemented in the start menu is a parallax effect that is activated when the player moves their mouse around the screen. This effect was achieved using the script DynamicStartScreen.cs which can be viewed in Appendix F (1.6). The script calculates the distance of each layer in the scene from its starting position and adjusts the parallax effect multiplier accordingly, creating a sense of depth. The mouse's coordinates on the screen are used to calculate each layer's horizontal and vertical movement, moving them towards their target positions using a smoothing effect that slows down the further away the mouse is from the centre of the screen.

In addition to the dynamic start screen, the CreditsMenu.cs script was created to handle the credits menu UI that appears and disappears when the player interacts with the credits button. This script utilises the isPaused variable to toggle the credits menu on and off and includes functionality to play audio effects when the player interacts with the menu. The CreditsMenu script can be viewed in Appendix F (1.5). Audio effects are significant here to give the user feedback when they interact with the menu items.


DEVELOPER: WILL DIBBLE

ADDITIONAL PACKAGES & ASSETS:
ARON GRANBERG'S A* PATHFINDING PROJECT
SPRITES ADAPTED FROM PIXEL FROG'S PIXEL ADVENTURE
ASSET PACKS
SOUND EFFECTS BY DAGURASU

Image 16: Credits

Lastly, I developed the StartMenu.cs script to define the behaviour of the start menu, including loading levels, resetting the time scale, and playing sounds upon interaction as the credits menu does. The StartMenu script can be viewed in Appendix F (1.23). This script handles the button functionality for starting the game, accessing the tutorial, and returning to the main menu. The corresponding level is loaded when a button is clicked and appropriate sound effects are played. The script also handles the destruction of the Timer object, something that will be implemented with the pause menu in a later build.

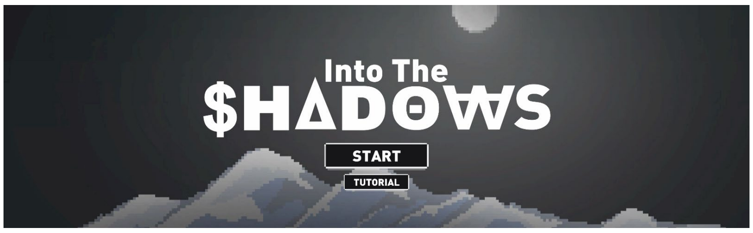


Image 17: Start Menu

The implementation of these features resulted in a functional and visually appealing start menu, effectively guiding the player through the game's various options and providing an effective user experience.

5.2.3. Create a Second Level with a Boss Fight Element (Objective 2.3)

To provide players with more of a challenging experience, I designed and implemented a second level featuring a boss fight. This encounter involves an indestructible creature that chases the player through the level. The player must use their platforming skills and utilise the invisibility feature to evade the pursuing enemy. It starts with the player emerging from the cave that concluded the previous scene. As the player exits the cave, a background trigger changes the parallax background to an outdoor setting, and the terrain becomes grassy.

Simultaneously, another trigger activates the enemy boss, which begins chasing the player. The player then faces a series of obstacles, including breakable platforms, moving platforms, sawblades, and combinations of these elements. After overcoming these challenges, the player enters another cave, marking the beginning of the third level.

The enemy boss's behaviour and movement are controlled by the 'EnemyBossMovement.cs' script. This script is attached to the enemy boss object and governs its movement towards the player and its attempt to catch up with and kill the player. The EnemyBossMovement script can be viewed in Appendix F (1.7).

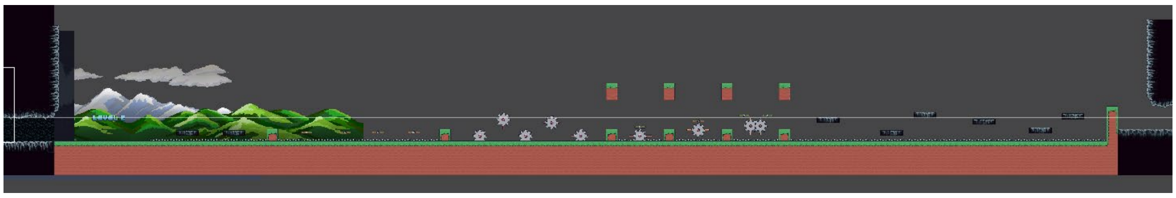


Image 18: Level 2 Complete

During the initialisation phase, the script retrieves necessary components, such as the rigid body, and starts the jump routine. The physics updates are handled in the FixedUpdate function, where the script checks if the boss has caught up to the player. If the boss is within a specified catch-up distance, the player's 'Die()' function is called, resulting in the player's death and the level restarts. The boss's vertical position is also limited in this function to prevent it from falling below the terrain. This is implemented as it needs to be able to pass through the landscape at specific points in the level whilst appearing as though it is touching the ground.

Image 19: Level 2 Boss Enemy

The Update function controls the boss's head rotation, ensuring that it is consistently looking at the player. This effect is achieved by calculating the angle between the boss's head and the player's position and updating the head's rotation accordingly. I felt this was an essential addition as it gives the player an uneasy feeling that the enemy is staring at them, increasing the pressure of the task. Lastly, the coroutine 'JumpRoutine()' manages the boss's jumping behaviour. The boss jumps towards the player at a frequency determined by the player's visibility state, with the boss moving more frequently when the player is visible. Doing so gives the player more time when invisible but makes the platforming aspects more challenging.

By incorporating these various elements and behaviours, the second level provides players with an exciting and challenging boss fight experience that tests their platforming skills and mastery of the invisibility feature.

5.2.4. Improve the AI for The Enemies (Objective 2.4)

I enhanced the AI of the game's enemies by adding dynamic behaviour based on the player's visibility status. This improvement included adjustments to patrolling, searching, and reacting behaviours for the rolling and jumping enemies depending on the player's visibility.

The rolling enemies' AI behaviours were adapted to account for the player's invisibility status. When the player is invisible, these enemies move slower as they become more cautious in their search for the player. This change was made to the RollingEnemyAI.cs script, which adjusts the enemy's speed based on the player's visibility status.

The jumping enemies also exhibit altered behaviour when the player is invisible. In this state, they lose the ability to detect the player and remain stationary, no longer pursuing the player. This change was implemented in the JumpingEnemyAI.cs script, which alters the enemy's pathfinding and movement.

5.2.5. Add Sound Effects for the Enemies and Environment (Objective 2.5)

Implementing sound effects for various game elements enhances the overall game experience and provides essential feedback to the player. In the RollingEnemyAI.cs and JumpingEnemyAI.cs scripts, sound effects are implemented to correspond with enemy movement and death states by employing the AudioManager component to manage the sound effects.

5.2.6. Implement a Timer Score System (Objective 2.6)

My implementation of the timer score system aimed to encourage players to complete levels more quickly and efficiently, adding a challenging aspect to the gameplay. To achieve this, I first designed a panel image in Photoshop, created a new UI object, and set the image within Unity. A text object was then positioned on top of this image to display the timer value.

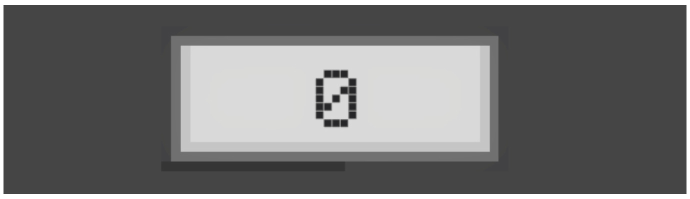


Image 20: Timer UI

The Timer.cs script defines the Timer class, which tracks the elapsed time during gameplay and manages bullet counts. The class contains functions for starting and stopping the timer and getters and setters for bullet counts and bullet level counts. Furthermore, it has functions for subscribing to and unsubscribing from the SceneManager's sceneLoaded event, which starts the timer when the Level 1 scene is loaded and stops the timer when the Final Score Screen scene is loaded. This script is attached to the timer object that persists from scene to scene.

The TimerUICounter.cs script, which can be found in Appendix F (1.25), is attached to the TimerUICounter text and updates the TextMeshProUGUI component of the TimerUI with the current time value, formatted as a whole number. The Timer script is located in the scene, and the current game time is assigned to a local variable. During each frame update, it retrieves the TextMeshProUGUI component attached to the game object and sets its text to the current time value from the Timer script.

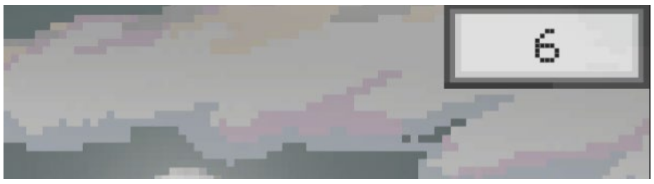


Image 21: Timer UI Implemented

Both scripts play crucial roles in the timer score system's functionality, providing players with a clear indication of their progress throughout each level. Incorporating this system into the game introduces a new layer of challenge and engagement, enhancing the player experience.

5.2.7. Implement a UI for the Bullet System (Objective 2.7)

For this objective, I focused on displaying the player's current bullet type, remaining ammunition, and available bullet types to create a more detailed and comprehensive UI for the bullet system. The UI design ensures that players have access to essential information about their combat resources, allowing them to switch between bullet types easily.

I designed a long panel similar to the timer counter UI in Photoshop and added three tabs below it. The tabs were coloured green, one at a time, to indicate the selected bullet type. I then exported all three colour variations for use in Unity. The panel displays the three bullet pickup images, accompanied by text numbers indicating the number of each bullet type the player possesses.

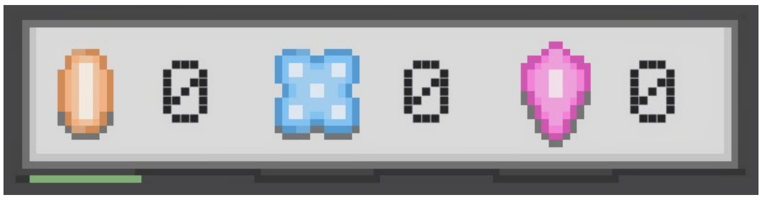


Image 22: Bullet UI

The UI implementation is primarily handled by the WeaponSelection script, which is attached to the player object. This script manages the player character's weapon selection, bullet counts, and weapon-related actions and includes methods for updating UI elements, such as the bullet count text and the currently selected bullet image.

The maximum number of each bullet type a player can hold is 9. The text colour for the bullet count changes to red when the player has the maximum number of bullets.

The UI image changes when the player presses the 'V' key to cycle between bullet types giving the illusion that the green bar is being used to do this. The Update method in the WeaponSelection script listens for the 'V' key press. When detected, the script increments the currentWeaponIndex and wraps it around to 0 if it exceeds the length of the weaponSprites array. The bulletBoxImage.sprite is then updated to display the sprite corresponding to the new currentWeaponIndex.

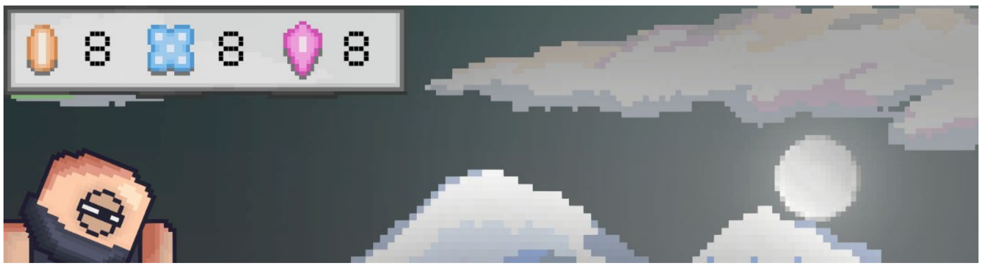


Image 23: Bullet UI Implemented

This implementation ensures that the bullet UI provides players with all the necessary information about their available ammunition and current bullet type while also offering a smooth and responsive interface for switching between bullet types during gameplay.

5.3. User Playtesting

5.3.1. Introduction

After developing Build 2, I conducted a playtesting session with six diverse participants to gather feedback on gameplay, mechanics, and overall experience, thereby identifying potential issues and areas of improvement for the development of Build 3.

5.3.2. In-depth Analysis of Participants' Feedback

Participants 1 and 4, both intermediate and advanced gamers, enjoyed the game and praised the invisibility feature for adding strategy. They suggested improving enemy AI and incorporating power-ups for a more engaging experience. Participant 4 also mentioned difficulty inconsistencies, recommending more variety in enemy movements and obstacles.

Participants 2 and 5, both beginners, found the game enjoyable but struggled with controls. They suggested making the invisibility state slightly more noticeable and improvements in the tutorial level to better explain controls. Participant 5, with no prior experience in 2D platformers, found the invisibility feature less necessary due to difficulties using it alongside other controls.

Participants 3 and 6, both intermediate gamers, provided valuable feedback on mechanics and difficulty. They recommended refining enemy control mechanisms, binding bullets to different buttons, and expanding AI capabilities. Participant 6 suggested adding a cooldown or energy meter for the invisibility feature and implementing a progression system with unlockable abilities or upgrades.

5.3.3. Common Themes and Conclusion

The feedback gathered from participants revealed several common themes, with the majority expressing appreciation for the game's invisibility feature while recommending enhancements to its visibility and mechanics. Additionally, they emphasised the need for increased variety in enemy types, improved controls, and the implementation of power-ups or progression systems. Addressing these concerns will help optimise Build 3, providing a more engaging and enjoyable gameplay experience.

Considering the invisibility feature, one potential improvement could be introducing a subtle shimmering effect while the player is invisible. However, implementing such an effect might reduce the gameplay's difficulty, which should be carefully considered. While the power-up and progression system would be an excellent addition, time constraints in developing Build 3 may prevent its implementation at this stage. Nonetheless, it could be considered for future updates.

Regarding enemy improvements, introducing a third enemy AI and a final boss level would both present a challenging and rewarding experience for players, offering a sense of accomplishment upon completion. The final enemy boss should be more demanding than any previous encounters in the game, ensuring players conclude their experience on a high note.

As for the bullet swapping system, the simplicity of the current controls will be maintained, despite suggestions for change. However, future updates may explore the possibility of a more complex control system, potentially integrated with the power-up and progression systems mentioned earlier.

5.3. Build 3 (Extra Functionality)

Build 3 required me to develop extra functionality for the game, such as new abilities, more challenging obstacles and enemies, enhanced visuals, music, and overall polish.

5.3.1. Implement a Third Enemy into Level 1 (Objective 3.1)

As influenced by the user testing, I introduced the third enemy type, a flying enemy that hovers above the player and drops TNT. The FlyingEnemyAI.cs script was created to implement these new AI behaviours, which handles the flying enemy's movements through the air and the dropping of the TNT blocks. The FlyingEnemyAI script can be viewed in Appendix F (1.9).

This enemy has distinct behaviours depending on the player's visibility status. When the player is visible, the flying enemy drops one TNT block. However, when the player is invisible, the flying enemy drops three TNT blocks: one central block and two additional blocks to the left and right of the central block. The TNT blocks are tagged as traps, causing player death upon contact. Doing so makes the enemy more difficult to combat whilst trying to hide from the jumping and rolling enemies, making the player think more strategically when using the power.

Image 23: Flying Enemy

Additionally, the flying enemy script plays a drop sound effect when a TNT block is created under the game object, and the TNT_Impact.cs script handles TNT box destruction and triggers an explosion sound effect when the box is destroyed. As before, the AudioManager component is used to achieve this integration.

5.3.2. Create a Third Level with a Double Jump Ability and More Challenging Obstacles (Objective 3.1)

In the third level, I introduced a double jump ability to enhance player movement and expand platforming mechanics. The level is set in a cave with breakable platforms, requiring players to navigate vertically using the new double jump ability. Players must use their invisibility power to reveal hidden platforms and avoid hazards, with invisible platforms, spikes, blades, and bat enemies becoming visible and interactable when the player turns invisible.

The vertical ascent challenges players to alternate between invisibility states to discover the path upwards and avoid dangerous hazards. Successfully navigating this section leads to loading the final level, Level 4. The script, InvisiblePlatform.cs (Appendix F, 1.10), manages the visibility and interactivity of platforms, spikes, blades, and bats in response to the player's invisibility state, updating the object's visibility, collider, and tag as necessary.

This level design offers a challenging experience with new mechanics requiring skilful navigation and quick decision-making. The double jump ability enhances platforming mechanics, while the invisibility interactions increase complexity and strategic elements. Rolling, jumping, and flying enemies are not implemented in this level.

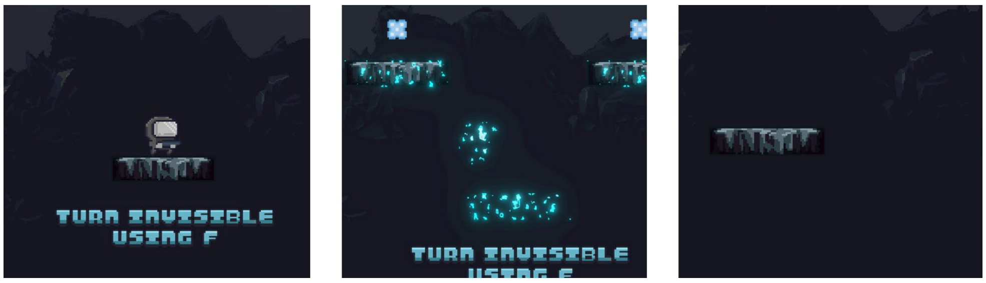


Image 24: Flip-Flopping Invisible Platforms

The double jump ability offers an engaging progression in the game's platforming mechanics, while the invisibility interactions increase the complexity and strategic elements of the level. Due to these new obstacles and strategy methods, the rolling, jumping, and flying enemies are not implemented in this level.

5.3.3. Create a Fourth and Final Level that has a Boss Fight and a Final Score Screen (Objective 3.2)

For level 4, I designed a boss fight featuring a large bat with A* pathfinding, similar to other enemies' AI in the game. The player navigates between platforms, avoiding spikes and using double jumps. Turning invisible reveals additional platforms and bullet pickups. The bat mimics the player's visibility with a one-second delay, and upon defeat, the player is directed to the ending screen. A "Bat Boss Health" bar at the top shows progress towards completing the level.

The script "BatEnemyAI.cs" governs the bat's behaviour, including pathfinding, health, speed, and visibility. It initialises component references and sets up a repeating InvokeRepeating() call to update the bat's path. The FixedUpdate() method moves the bat towards the player and adjusts its facing direction.

The HandleInvisibility() method, called in the Update() method, manages the bat's response to the player's invisibility status. The TakeDamage() method reduces the bat's health and updates the health bar. If the health reaches zero, the Die() method destroys the bat and transitions to the "Final Score Screen" scene. The BatEnemyAI script can be viewed in Appendix F (1.2).

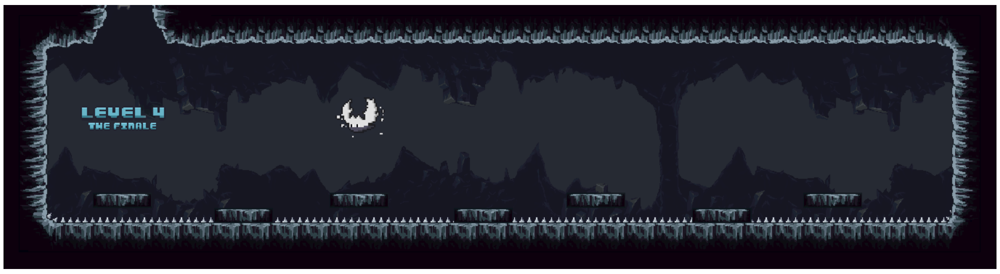


Image 24: Level 4 Complete

5.3.4. Implement a Visual Aesthetic When Changing Levels (Objective 3.4)

To enhance the visual aesthetics during level transitions, I implemented a fading UI screen that smooths the transition between levels. This process involved creating a script for managing a UI image's fade-in and fade-out process and a script for managing level loading and transition effects.

The first script, TransitionImageController.cs, handles the fade-in and fade-out processes. It initialises the relevant components and includes two methods, FadeIn() and FadeOut(), which call a private IEnumerator method Fade() to handle the processes. The Fade() method adjusts the image's alpha value throughout the transition time incrementally and invokes an onComplete action if provided.

The second script is the LevelManager script. When any scene is loaded, it calls the FadeOut() method from the TransitionImageController to perform the fade-out effect. It initiates the fade-in transition and waits for it to complete before loading the specified level using the SceneManager.LoadScene() method.

Implementing these scripts makes the transitions between levels look much smoother and less abrupt.

5.3.5. Incorporate Music into the Game (Objective 3.5)

To enhance the player experience and create a distinct atmosphere for each level, I integrated various background music tracks for the game's different levels, menu, and victory screens. I integrated background music tracks for different levels, menu, and victory screens, using the AudioManager script to manage audio and maintain player immersion. The track "Relaxing" by Music For Videos is used for the start, tutorial, and ending screens (Music For Videos, 2023). "Dreamy Space Soundtrack" by Cactusdude features on Levels 1, 2, and 3 (Cactusdude, 2021).

For the Level 4 boss fight, "Good Day To Die" by Miguel Johnson is used (Johnson, 2018), combined with "Large Wings Flapping Foley" (ToTrec2, 2021) and a "monster roar" sound effect from Mixkit (Mixkit, 2023). These additional sound effects create a more immersive experience during the climactic level.

The AudioManager script ensures appropriate music is played for each scene, delivering a rich audio experience complementing the gameplay and setting.

5.3.6. Polish and Fine Tune the Game (Objective 3.6)

I dedicated time to polishing and fine-tuning the game, addressing any remaining bugs, balancing gameplay elements, and ensuring that all visual and audio assets were polished and cohesive. This process ensured the final product was high quality and provided players with a satisfying and enjoyable experience.

6. Conclusions and Discussion

This chapter synthesises and evaluates the project's success in achieving its objectives. It addresses research questions regarding the 2D platformer market innovation and reflects on the project's contributions and personal growth during development.

6.1 Objective Fulfilment and Conclusions

The project aimed to create a unique 2D platformer with innovative features like player invisibility and a bullet-collecting system. The project met its objectives through iterative development and player testing, offering a fresh, engaging experience.

6.2 Discussion of Conclusions

The game's distinct mechanics and advanced enemy AI contribute to the genre's evolution, aligning with academic literature recommendations. The invisibility and bullet systems set this platformer apart, keeping players engaged and demonstrating the project's impact on the 2D platformer design field.

6.3 Implications for Future Work

Future projects could explore advanced power-up systems, complex control schemes, and diverse obstacles or enemy types and incorporate educational elements to enhance players' depth, replayability, and value.

6.4 Personal Progress and Reflection

The project exhibited effective management and adaptability, leading to the game's success. Insights gained from game design, programming, and player feedback align with academic findings, emphasising the importance of mechanics, level design, visuals, and audio in crafting immersive experiences. With hindsight, more detailed initial planning and research into power-up systems would be considered.

In conclusion, the project achieved its objectives, contributed to the 2D platformer design field, and offered valuable insights for future work. Personal growth and challenges throughout the project led to a deeper understanding of game development, management, and the importance of player feedback in creating engaging experiences.

Reference List

Bibliography

A. Khalifa, F. de Mesentier Silva and J. Togelius. (2019). “Level Design Patterns in 2D Games”. IEEE Conference on Games (CoG), London, UK, 2019, pp. 1-8, doi: 10.1109/CIG.2019.8847953.

Adams, E. (2014). Fundamentals of Game Design. New Riders. 9780133812329

Blow, J. (2004). Game Development: Harder Than You Think. ACM Queue, 1(10), 28-37. DOI: 10.1145/971564.971590

Brackeys. (2020). Get started with 2D Shader Graph in Unity - Dissolve Tutorial. [Video]. https://www.youtube.com/watch?v=5dzGj9k8Qy8

Cactusdude. (2021). Dreamy Space Soundtrack. Itch.io. https://chiphead64.itch.io/dreamy-space-soundtrack

Coding in Flow (2021). Build a 2D Platformer Game in Unity | Unity Beginner Tutorial. [Video]. https://youtube.com/playlist?list=PLrnPJCHvNZuCVTz6lvhR81nnaf1a-b67U

Collins, K. (2013). The Game Audio Tutorial: A Practical Guide to Sound and Music for Interactive Games. CRC Press. 9781136127014

Cook, D. (2006). What are game mechanics? Lostgarden. https://lostgarden.home.blog/2006/10/24/what-are-game-mechanics/

Dagurasu. (2021). Retro Sounds. Itch.io. https://dagurasusketch.itch.io/retrosounds

Gee, J.P. (2003). What Video Games Have to Teach Us About Learning and Literacy. Association for Computing Machinery. https://doi.org/10.1145/950566.950595

Granberg, A. (2021). A* Pathfinding Project. AronGranberg.com. https://arongranberg.com/astar/download

JDWasabi. (2021). 8-bit / 16-bit Sound Effects (x25) Pack. Itch.io. https://jdwasabi.itch.io/8-bit-16-bit-sound-effects-pack

Johnson, M. (2018). Good Day To Die by Miguel Johnson. breakingcopyright.com. https://breakingcopyright.com/song/miguel-johnson-good-day-to-die

Kasurinen, J. and Knutas, A. (2018). “Publication trends in gamification: A systematic mapping study,” Computer Science Review, 27, pp. 33–44. Available at: https://doi.org/10.1016/j.cosrev.2017.10.003.

Mixkit. (2023). Free Monster Sound Effects. Mixkit.co. https://mixkit.co/free-sound-effects/monster/

Music For Videos. (2023). Relaxing. Pixabay.com. https://pixabay.com/music/modern-classical-relaxing-145038/

Pav. (2020). Parallax Scrolling in pixel-perfect 2D Unity games. PavCreations. https://pavcreations.com/parallax-scrolling-in-pixel-perfect-2d-unity-games/

Rogers, S. (2014). Level Up! The Guide to Great Video Game Design. Wiley. 9781118877166

Schell, J. (2008). The Art of Game Design: A Book of Lenses. CRC Press. 9780123694966

Thorsen, T. (2010). New Super Mario Bros. Wii sales near 10.5 million, GameSpot. GameSpot. Available at: https://www.gamespot.com/articles/new-super-mario-bros-wii-sales-near-105-million/1100-6246941/ (Accessed: May 2, 2023).

ToThrec2. (2021). Large Wings Flapping – Foley. Freesound.com. https://freesound.org/people/tothrec2/sounds/596541/

Unity Technologies. (2021). Unity User Manual (2021.1). https://docs.unity3d.com/Manual/index.html

Wood, A. (2017). Cuphead breaks two million copies sold, over half of them on steam, pcgamer. PC Gamer. Available at: https://www.pcgamer.com/cuphead-breaks-two-million-copies-sold-over-half-of-them-on-steam/

Yin-Poole, W. (2012). Super Meat Boy sells over 1 million. Eurogamer.net. Retrieved from http://www.eurogamer.net/articles/2012-01-03-supermeat-boy-sells-over-1-million

Resources

Backgrounds: Located at "Assets/My Assets/Background/Parallax" folder.

File names: 1.png, 2png, 3.png, 4.png.

Author: William Dibble & OlegKrylov

Downloaded at: Created by William Dibble, developed from assets found at https://opengameart.org/content/pixel-art-parallax-background (Creative Commons 4.0 License)

Accessed: 02/03/2023

Cave Platforms: Located at "Assets/My Assets/Cave Platforms" folder.

File names: Cave-Platform-1.png, Cave-Platform-1-Falling.png

Author: William Dibble & Szadiart

Downloaded at: Created by William Dibble, developed from assets found at https://szadiart.itch.io/pixel-fantasy-caves (Royalty Free Assets)

Accessed: 02/03/2023

Bat Boss Enemy: Located at "Assets/My Assets/Enemies/Bat" folder.

File names: Bat (48 x 384).png, Bat Masked (48 x 384).png

Author: William Dibble

Downloaded at: Created by William Dibble

Accessed: 02/03/2023

Jumping Enemy: Located at "Assets/My Assets/Enemies/Big Guy" folder.

File name: Idle (32x32).png

Author: William Dibble

Downloaded at: Created by William Dibble

Accessed: 02/03/2023

Level 2 Boss: Located at "Assets/My Assets/Enemies/Boss1" folder.

File names: Boss Idle (64 x 768).png, Boss Idle HEAD (27 x 840).png, Boss Idle.png

Author: William Dibble

Downloaded at: Created by William Dibble

Accessed: 02/03/2023

Flying Enemy and TNT: Located at "Assets/My Assets/Enemies/FlyingEnemy” folder.

File names: FlyingEnemy_Box.png, FlyingEnemy_Box_Impact.png, FlyingEnemy_Idle (32x32).png

Author: William Dibble

Downloaded at: Created by William Dibble

Accessed: 02/03/2023

Rolling Enemy: Located at "Assets/My Assets/Enemies/RollingEnemy" folder.

File name: RollingEnemy_Idle (32x32).png

Author: William Dibble

Downloaded at: Created by William Dibble

Accessed: 02/03/2023

Fonts: Located at "Assets/My Assets/Fonts" folder.

File name: PublicPixel.ttf

Author: GGBotNet

Downloaded at: https://www.dafont.com/public-pixel.font (Creative Commons Zero v1.0 Universal License)

Accessed: 02/03/2023

Bullet Shots: Located at "Assets/My Assets/Main Character/Gun" folder.

File names: Shot1.png, Shot2.png, Shot3.png, Shot1-destroy.png, Shot2-destroy.png, Shot3-destroy.png, Shot1-Pickup-Animation.png, Shot2-Pickup-Animation.png, Shot3-Pickup-Animation.png

Author: William Dibble

Downloaded at: Created by William Dibble

Accessed: 02/03/2023

Main Character Sprite: Located at "Assets/My Assets/Main Character" folder.

File names: Appearing (96x96).png, Disappearing (96x96).png, Fall (32x32).png, Hit (32x32).png, Idle (32x32).png, Jump (32x32).png, Run (32x32).png.

Author: William Dibble & Pixel Frog

Downloaded at: Created by William Dibble, developed from assets found at https://pixelfrog-assets.itch.io/pixel-adventure-1 (Public Domain)

Accessed: 02/03/2023

Boss Fight Music: Located at "Assets/My Assets/Music" folder.

File name: Boss fight.mp3

Author: Miguel Johnson

Downloaded at: https://breakingcopyright.com/song/miguel-johnson-good-day-to-die (CC License 3.0)

Accessed: 02/03/2023

Level 1, 2, & 3 Music: Located at "Assets/My Assets/Music" folder.

File name: Music 1.mp3, Music 2.mp3

Author: Cactusdude

Downloaded at: https://chiphead64.itch.io/dreamy-space-soundtrack (CC License 4.0)

Accessed: 02/03/2023

Menu & Tutorial Music: Located at "Assets/My Assets/Music" folder.

File name: Screen Music.mp3

Author: Music For Videos

Downloaded at: https://pixabay.com/music/modern-classical-relaxing-145038/ (Royalty Free)

Accessed: 02/03/2023

Blade Texture: Located at "Assets/My Assets/Obstacles/Blade" folder.

File name: Blade (38x38).png

Author: Pixel Frog

Downloaded at: https://pixelfrog-assets.itch.io/pixel-adventure-1 (Public Domain)

Accessed: 02/03/2023

Spike Texture: Located at "Assets/My Assets/Obstacles/Spikes" folder.

File name: Idle.png

Author: Pixel Frog

Downloaded at: https://pixelfrog-assets.itch.io/pixel-adventure-1 (Public Domain)

Accessed: 02/03/2023

Moving Platform Texture: Located at "Assets/My Assets/Platforms" folder.

File name: Wood (32x8).png

Author: Pixel Frog

Downloaded at: https://pixelfrog-assets.itch.io/pixel-adventure-1 (Public Domain)

Accessed: 02/03/2023

Dissolve Material: Located at "Assets/My Assets/Shaders" folder.

File name: Dissolve.mat

Author: William Dibble

Downloaded at: Created by William Dibble

Accessed: 02/03/2023

Sound Effects: Located at "Assets/My Assets/Sound Effects" folder.

File names: bigJump.wav, blow.wav, cinematic1.wav, cinematic2.wav, coin1.wav, coin2.wav, collect.wav, curious.wav, dot.wav, explosion.wav, fall_impact.wav, fire.wav, hit1.wav, hit2.wav, hit3.wav, invisible1.wav, invisible2.wav, jump1.wav, jump2.wav, levelup.wav, machine1.wav, question.wav, Select 1.wav, select1.wav, select2.wav, select3.wav, select4.wav, slide_down.wav, Text 1.wav, transition.wav

Authors: Dagurasu / JDWasabi / William Dibble

Downloaded at: https://jdwasabi.itch.io/8-bit-16-bit-sound-effects-pack & https://dagurasusketch.itch.io/retrosounds (Royalty Free) (Some sound are edited and adapted by William Dibble)

Accessed: 02/03/2023

Terrain 1.1 Textures: Located at "Assets/My Assets/Terrain" folder.

File name: Terrain1.png

Author: William Dibble & Pixel Frog

Downloaded at: Created by William Dibble, developed from textures by Pixel Frog https://pixelfrog-assets.itch.io/pixel-adventure-1 (Public Domain)

Terrain 1.2 Textures: Located at "Assets/My Assets/Terrain 2" folder.

File name: Terrain (16x16) 1.png

Author: William Dibble & Pixel Frog

Downloaded at: Created by William Dibble, developed from textures by Pixel Frog https://pixelfrog-assets.itch.io/pixel-adventure-1 (Public Domain)

Accessed: 02/03/2023

Cave Terrain Textures and Background: Located at "Assets/My Assets/Terrain 3 - Cave" folder.

File names: Cave 2 (16x16).png, background3.png

Author: William Dibble & Szadiart

Downloaded at: Created by William Dibble, developed from assets found at https://szadiart.itch.io/pixel-fantasy-caves (Royalty Free)

Accessed: 02/03/2023

Pixel Art Text Textures: Located at "Assets/My Assets/Text" folder.

File name: ABC.png

Author: William Dibble & Pixel Frog

Downloaded at: Created by William Dibble, developed from textures by Pixel Frog https://pixelfrog-assets.itch.io/pixel-adventure-1 (Public Domain)

Accessed: 02/03/2023

UI Files: Located at "Assets/My Assets/UI" folder.

File names: BatEnemyHealthUI.png, BulletUI-Shot1.png, BulletUI-Shot2.png, BulletUI-Shot3.png, Credits Button.png, FadeBlackScreen.png, ITS Logo White.png, ITS Logo.png, Main Menu Button.png, PauseMenuUI.png, Resume Button.png, Start Button.png, TimerUI-PNG.png, Turorial Button.png, Vignette.png

Author: William Dibble

Downloaded at: All UI elements created by William Dibble

Accessed: 02/03/2023

Glossary

Unity Jargon

Scene

A Scene in Unity is a container that holds all the game objects, assets, and environments for a particular level or section of the game. It represents a 2D or 3D world in which game objects can be placed, interact with each other, and perform various actions.

GameObject

A GameObject is the fundamental object type in Unity. It represents any entity within the game world, such as characters, props, or environment elements. GameObjects can have various components added to them to define their behaviour, appearance, and interactions.

Prefab

A Prefab is a reusable, customisable template for a GameObject or a group of GameObjects. It allows developers to create, configure, and store a GameObject with all its components and properties, which can then be instantiated multiple times throughout the game. This helps streamline development and maintain consistency across the game.

Script Component

A Script Component is a custom script written in C# that can be attached to a GameObject to define its behaviour, control its properties, and interact with other GameObjects or game systems. Script Components allow developers to create logic and functionality for their games.

Audio Source Component

The Audio Source Component is a component that can be attached to a GameObject to play sounds and control various audio settings like volume, pitch, and looping. It allows developers to add audio effects and background music to enhance the game experience.

Animator Component

The Animator Component is responsible for controlling and managing animations on a GameObject. It works with Unity's animation system to blend, transition, and control animations based on defined parameters or user input.

Inspector

The Inspector is a panel in the Unity editor that displays and allows editing of the properties and components of a selected GameObject or asset. It provides an interface for developers to configure and fine-tune their game objects and assets.

Rigidbody 2D Component

The Rigidbody 2D Component adds physics-based behaviour to a 2D GameObject, such as gravity, collision detection, and forces. It allows developers to create realistic movements and interactions between 2D objects within the game world.

Sprite Renderer Component

The Sprite Renderer Component renders 2D images (sprites) on GameObjects in a 2D game. It allows developers to control properties like sorting layers, transparency, and colour tints to create visually appealing 2D games.

Box Collider 2D Component

The Box Collider 2D Component is a 2D physics collider that can be attached to a GameObject to detect collisions and trigger events. It defines an invisible rectangular bounding box interacting with other colliders in the game world to enable physics-based interactions.

Scene Management

Scene Management refers to organising, loading, and transitioning between different scenes within a game. It involves creating a structure to manage the game's flow, allowing developers to control which scenes are active, unloaded, or loaded during gameplay.

Appendices

All appendices have been submitted in the report.

Appendix A: Project Definition Document

Problem Description

Recent years have seen an oversaturation of 2D platformers in the gaming industry. Many games within this genre are similar in their gameplay and features. My goal is to create a unique game that stands out in the market by implementing new features that haven't been used in a modern 2D platformer before.

My feature concept would add the ability for the player to turn invisible on command, allowing them to sneak past enemies without detection. The downside to the feature would be that it impairs the player's vision, making dodging dangerous obstacles harder.

Additionally, the game will also include a bullet collecting and switching system, where the player can collect different types of bullets with varying properties and switch between them in combat. This game will take inspiration from other successful 2D platformers such as Super Meat Boy, Thomas Was Alone, Ori, Rayman, and Super Mario.

Objectives

The main objective of this project is to create a unique and engaging 2D platformer game that stands out in a saturated market by implementing new features, such as the ability for the player to turn invisible, a bullet collecting and switching system, and advanced AI for the enemies.

Sub Objectives:

- Implement a game mechanic for the player to activate invisibility and create a visual impairment effect to add difficulty that outweighs the advantages of invisibility.
- Implement a system for the player to pick up and use different types of bullets in combat. This includes creating various types of bullets with different properties, and a mechanic for the player to collect and switch between them.
- Create levels and enemies that are challenging but fair to the player. This includes designing levels with obstacles and enemies that provide a challenge to the player but are not impossible to overcome.
- Create a story and visual aesthetic that is engaging and immersive, that includes developing a narrative for the game, and creating artwork and sound effects that are visually pleasing and enhance the player's experience.

- Create enemies with advanced and dynamic AI, which change their behaviour depending on whether the player is invisible or not.
- Test the game with a group of players to gather feedback and make improvements. This will include arranging playtesting sessions with players that have varying levels of experience with video games and gathering feedback on the gameplay, level design, art, and sound. Using this feedback, I can make improvements to the game before release. I will need to research different methods of playtesting and gathering feedback in game development.

Design

To incorporate the iterative design process in my final project, I will break down the game development sub-objectives above into three stages. This methodical approach will allow me to systematically improve the game at each stage, ensuring the final product is of the highest quality.

Build 1: Basic Functionality

In the first stage, I will focus on establishing the foundation of the game by implementing the core mechanics such as the invisibility feature and visual impairment effect. This will involve experimenting with different methods of implementing invisibility and visual impairment effects within Unity. I will also be creating a prototype of the game, which will serve as a proof of concept for the core mechanics. This will include creating a basic player character with the invisibility ability and implementing a simple level for testing the mechanics. Additionally, I will also be setting up the basic visual structure for the game, including the user interface and menu systems.

Build 2: Main Functionality

In the second stage, I will continue to develop the gameplay elements such as the bullet collecting and switching system, creating challenging and well-balanced levels, and designing enemies that have advanced dynamic AI that changes depending on whether the player is invisible or not. I will also be creating a variety of different bullets with various properties, such as bouncing bullets and sticky bullets, to give the player more options in combat. Additionally, I will be creating a variety of levels that are challenging but fair to the player and I will be designing enemies that are unique and engaging.

Build 3: Extra Functionality

The final stage will be dedicated to polishing the game, which includes creating a compelling story and immersive visual aesthetics. This will involve writing and implementing a narrative for the game, creating detailed character designs, and developing a cohesive visual style for the game. I will also incorporate a system for the player to upgrade their abilities and weapons, which will add an extra layer of depth to the gameplay and additionally, I will be incorporating a system for the player to customise their character, such as changing their appearance, which will give players more options to personalise their experience.

Finally, I will be creating a quality assurance plan, to ensure that the game is thoroughly tested and that any bugs or issues are identified and addressed before release. This will involve setting up a dedicated QA system, creating a testing schedule, and creating a process for reporting and tracking bugs and issues.

Beneficiaries

The expected beneficiaries of this project include:

- Myself, as the developer of this project, as it will provide learning opportunities in game development and the game industry.
- Players who are looking for new and unique 2D platformer games to play, as this game would provide them with a new gaming experience that they can enjoy.
- The 2D platformer game industry, as it would bring a new and unique game that adds to the variety of games available in the market.
- The game development community, as it will contribute to the knowledge base of game development with the implementation of unique features such as the invisibility ability mentioned.
- Potentially, it could also have educational benefits for young players by incorporating educational elements such as problem-solving and critical thinking. This game could be a fun and interactive way to teach problem-solving skills and critical thinking.
- Additionally, by potentially incorporating educational elements such as problem-solving and critical thinking, the game could also have potential educational benefits for young players.

Assumptions and Limitations

Assumptions made for this project include:

- My personal experience with Unity and 2D platformer game development will be sufficient to overcome most of the potential difficulties faced during this project. This includes my ability to code in C# and use Unity's engine, as well as my understanding of the basics of 2D platformer game development such as player movement, collision detection, and level design.
- The resources available online for Unity and C# game development will be sufficient to complete the project to a high standard. This includes tutorials, documentation, and

forums that can provide guidance and solutions to any problems that may arise during development.

- There will be a sufficient number of players interested in playing a 2D platformer game with my unique features. This includes players who are fans of 2D platformer games and are interested in unique gameplay mechanics that haven't been seen in modern platformer games before.
- Access to playtesters is possible, as it's important to have a group of players test the game and provide feedback. This includes recruiting playtesters from online communities, friends, or family.

Limitations for this project include:

- The scope of the project may need to be limited due to time and resource constraints. This includes potentially cutting certain features or levels to meet a specific deadline.
- The invisibility feature may prove to be too difficult to implement or may not be well-received by players. This includes the potential for technical difficulties in coding the feature or player feedback indicating it is not enjoyable or doesn't add enough to the game to keep it interesting.
- The game development and testing process may require more time and resources than anticipated. This includes the potential for unexpected bugs or issues to arise during development or testing that can prolong the process.

Risks

The general risks within this project include:

Objective	Likelihood	Severity	Score	Risks	Actions
General Risks					
General Risk 1 - Lack of Resources.	2	5	10	The game development may be hindered by a lack of funding, a lack of skill in Unity and 2D platformer game development, or a lack of necessary equipment or software.	To combat this risk, I will be using cost-effective solutions, such as using open-source software and royalty-free assets.
General Risk 2 - Changes in the Project Scope.	3	5	15	I may decide to change the requirements of the game, such as important features, at key stages in development, which may affect the timeline and outcome of the project.	I will ensure that my requirements are being met at each stage of development, and to address any concerns or changes in a timely manner. I will also have a clear and detailed plan that outlines the scope and objectives of the project, making it easier to manage changes in requirements.

General Risk 3 - Technology Failures.	2	5	10	The project may face serious failures in the technology, such as programming problems with the invisibility feature or bullet system that cannot be resolved, hardware failures, or incorrect assumptions about the feasibility of certain features.	To minimise this risk, I can research and test the technology being used before starting the project. I can also have a backup plan in place in case of technical failures, such as having a backup system or a plan to switch to alternative technology.
General Risk 4 - Difficulty in Implementing the Invisibility Feature.	2	4	8	The invisibility feature may prove to be too difficult to implement or may not be well-received by players, affecting the uniqueness and appeal of the game.	To combat this risk, I can test the invisibility feature thoroughly during development and gather feedback from play testers to ensure that it is a well-received and unique feature. I can also consider alternative features that may be easier to implement and still provide a unique experience.
General Risk 5 - Difficulty in Balancing the Game.	3	3	9	Difficulty in balancing the game's difficulty and making it challenging but fair for players, affecting the player's enjoyment and engagement with the game.	To minimise this risk, I can test the game at different difficulty levels and gather feedback from play testers to ensure that it is challenging but fair for players. I can also consider adjusting the difficulty level during development based on playtester feedback.
General Risk 6 - Difficulty in Creating an Engaging Story and Aesthetic.	4	3	12	Difficulty in creating an engaging story and aesthetic that immerses the player, affecting the player's enjoyment and engagement with the game.	To combat this risk, I can research successful 2D platformer games and take inspiration from their story and aesthetics. I can also gather feedback from the playtester to ensure that the story and aesthetic add effectively to the game.
General Risk 7 – Issues when Integrating an Advanced and Dynamic AI for the Enemies				1. Lack of diversity in the enemy behaviour. 2. Over complex AI behaviour that is complex and hard to debug. 3. Issues when balancing difficulty within the enemies.	1. I can implement methods to generate varied AI behaviour such as randomisation, dynamic decision-making, and machine learning techniques like reinforcement learning. Additionally, I can provide multiple AI personalities or behaviours for the enemies to choose from, increasing the variety of gameplay. 2. It will be important to simplify the AI behaviour and create clear and concise rules for how the AI should respond to different situations. I will be breaking down the AI behaviour into smaller, manageable parts and testing each part thoroughly before integrating them into the larger AI system. 3. To overcome this challenge, playtesting will be crucial. Regularly testing the game with different players will help to identify areas where the AI is too difficult or too easy and will allow me to adjust the AI behaviour accordingly.
General Risk 7 - Difficulty in Gathering the Playtester Feedback.	4	5	20	Difficulty in arranging playtesting sessions with a diverse group of players and gathering enough feedback to make improvements to the game before release.	I will plan playtesting sessions well in advance and actively seek out a diverse group of playtesters with varying levels of experience with video games. I will also have a structured feedback form or survey in place to gather information on different aspects of the game.

0-8: Low Risk	Likelihood: 1 (Very Unlikely) - 5 (Very Likely)

9-16: Medium Risk	Severity: 1 (Negligible) - 5 (Very Impactful)
17-25: High Risk	Score = Likelihood x Severity

Work Plan

WILLIAM DIBBLE - FINAL PROJECT		16 Jan	23 Jan	30 Jan	6 Feb	13 Feb	20 Feb	27 Feb	6 Mar	13 Mar	20 Mar	27 Mar	3 Apr	10 Apr	17 Apr	24 Apr	1 May	8 May
TASK	WEEK	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Project Definition Document																		
PDD Question Research	1	🟪																
PDD Risks Sheet	2		🟪															
PDD Gant Chart	2		🟪															
PDD Research Ethics Checklist	2		🟪															
PDD Self Review and Amends	3			🟪														
PDD Consultant Review	4				🟪													
PDD Second Review and Amends	4				🟪													
Final Project Report																		
Chapter 1: Introduction	5					🟪												
Chapter 2: Output Summary	6													🟪				
Chapter 3: Literature Review	6						🟪	🟪										
Chapter 4: Method	7								🟪	🟪	🟪	🟪	🟪					
Chapter 5: Results	12												🟪					
Conclusion / Discussion	13													🟪				
Project Review and Amends	14														🟪	🟪		
Project Submission	16																🟪	
Game Build 1: Basic Functionality																		
Create Terrain with Collision Detection	3			🟩	🟩													
Implement a Player with Movement	3			🟩														
Implement a Camera Follow for The Player	3			🟩														
Add Basic Sound Effects for The Player	3			🟩														
Design Textures for Terrain and Player	4			🟩	🟩	🟩	🟩											
Implement Invisibility Feature for The Player	4				🟩													
Create Dangerous Obstacles with Collision	4				🟩													
Create Enemies with Collision	4				🟩													
Create a Simple AI for the Enemies	5					🟩												
Create a Respawn System for The Player	5					🟩												
Implement a Level Start and a Finish	5					🟩												
Implement Bullet Collecting and Switching	6						🟩											
Testing and Bug Fixing	6						🟩	🟩	🟩									
Gather Feedback from Playtesting Sessions	6						🟩	🟩										
Game Build 2: Main Functionality																		
Create a Tutorial for The Player	7							🟩										
Create a Menu for The Player to Navigate	7							🟩										
Add a Boss Fight to The Game	7							🟩										
Create a Pause Menu with Settings	8								🟩									
Improve the AI for the Enemies	8								🟩									
Add Sound Effects (Enemies / Environment)	9									🟩								
Implement a Timer Score System	9									🟩								
Implement a UI for The Game	9									🟩								
Testing and Bug Fixing	10										🟩	🟩	🟩					
Gather Feedback from Playtesting Sessions	10										🟩	🟩						
Game Build 3: Extra Functionality																		
Create a Second Level with Double Jump Ability	11											🟩						
Create More Challenging Obstacles	11											🟩						
Create More Challenging Enemies	11												🟩					
Implement a Story Element	12												🟩					
Implement a Visual Aesthetic (Particles / Camera)	12													🟩				
Add Music to the Game	13													🟩				
Polish and Fine Tune the Game	13													🟩				
Testing and Bug Fixing	14													🟩	🟩			
Gather Feedback from Playtesting Sessions	14														🟩			

References

Aversa, D. (2022) *Unity Artificial Intelligence Programming: Add powerful, believable, and Fun AI Entities in your game with the Power of Unity.* Birmingham: Packt Publishing.

Burgun, K. (2012) *Game Design Theory, A New Philosophy for Understanding Games.* CRC Press.

Buttfield-Addison, P., Manning, J. and Nugent, T. (2019) *Unity Game Development Cookbook: Essentials for Every Game.* Beijing: O'Reilly Media.

Ferrone, H. (2019) *Learning C# by Developing Games with Unity 2019, Code in C# and Build 3D Games with Unity, 4th Edition.* Packt Publishing.

Godbold, A. and Jackson, S. (2016) *Mastering Unity 2D Game Development, Using Unity 5 to Develop a Retro RPG.* Packt Publishing.

Nystrom, R. (2014) *Game Programming Patterns.* Genever | Benning.

Pereira, V. (2014) *Learning Unity 2D Game Development by Example.* Packt Publishing.

Schell, J. (2014) *The Art of Game Design, A Book of Lenses, Second Edition.* Taylor & Francis.

Research Ethics Review Form

BSc, MSc and MA Projects

Computer Science Research Ethics Committee (CSREC)

http://www.city.ac.uk/department-computer-science/research-ethics

Undergraduate and postgraduate students undertaking their final project in the Department of Computer Science are required to consider the ethics of their project work and to ensure that it complies with research ethics guidelines. In some cases, a project will need approval from an ethics committee before it can proceed. Usually, but not always, this will be because the student is involving other people ("participants") in the project.

In order to ensure that appropriate consideration is given to ethical issues, all students must complete this form and attach it to their project proposal document. There are two parts:

PART A: Ethics Checklist. All students must complete this part.
The checklist identifies whether the project requires ethical approval and, if so, where to apply for approval.

PART B: Ethics Proportionate Review Form. Students who have answered "no" to all questions in A1, A2 and A3 and "yes" to question 4 in A4 in the ethics checklist must complete this part. The project supervisor has delegated authority to provide approval in such cases that are considered to involve MINIMAL risk.
The approval may be ***provisional*** – *identifying the planned research as* likely to involve MINIMAL RISK.
In such cases you must additionally seek ***full approval*** from the supervisor as the project progresses and details are established. ***Full approval*** must be acquired in writing, before beginning the planned research.

A.1 If you answer YES to any of the questions in this block, you must apply to an appropriate external ethics committee for approval and log this approval as an External Application through Research Ethics Online - https://ethics.city.ac.uk/		*Delete as appropriate*
1.1	Does your research require approval from the National Research Ethics Service (NRES)? *e.g. because you are recruiting current NHS patients or staff?* *If you are unsure try - https://www.hra.nhs.uk/approvals-amendments/what-approvals-do-i-need/*	**NO**
1.2	Will you recruit participants who fall under the auspices of the Mental Capacity Act? *Such research needs to be approved by an external ethics committee such as NRES or the Social Care Research Ethics Committee - http://www.scie.org.uk/research/ethics-committee/*	**NO**
1.3	Will you recruit any participants who are currently under the auspices of the Criminal Justice System, for example, but not limited to, people on remand, prisoners and those on probation? *Such research needs to be authorised by the ethics approval system of the National Offender Management Service.*	**NO**
A.2 If you answer YES to any of the questions in this block, then unless you are applying to an external ethics committee, you must apply for approval from the Senate Research Ethics Committee (SREC) through Research Ethics Online - https://ethics.city.ac.uk/		*Delete as appropriate*
2.1	Does your research involve participants who are unable to give informed consent? *For example, but not limited to, people who may have a degree of learning disability or mental health problem, that means they are unable to make an informed decision on their own behalf.*	**NO**

2.2	Is there a risk that your research might lead to disclosures from participants concerning their involvement in illegal activities?	**NO**
2.3	Is there a risk that obscene and or illegal material may need to be accessed for your research study (including online content and other material)?	**NO**
2.4	Does your project involve participants disclosing information about special category or sensitive subjects? *For example, but not limited to: racial or ethnic origin; political opinions; religious beliefs; trade union membership; physical or mental health; sexual life; criminal offences and proceedings*	**NO**
2.5	Does your research involve you travelling to another country outside of the UK, where the Foreign & Commonwealth Office has issued a travel warning that affects the area in which you will study? *Please check the latest guidance from the FCO - http://www.fco.gov.uk/en/*	**NO**
2.6	Does your research involve invasive or intrusive procedures? *These may include, but are not limited to, electrical stimulation, heat, cold or bruising.*	**NO**
2.7	Does your research involve animals?	**NO**
2.8	Does your research involve the administration of drugs, placebos or other substances to study participants?	**NO**
A.3 If you answer YES to any of the questions in this block, then unless you are applying to an external ethics committee or the SREC, you must apply for approval from the Computer Science Research Ethics Committee (CSREC) through Research Ethics Online - https://ethics.city.ac.uk/ **Depending on the level of risk associated with your application, it may be referred to the Senate Research Ethics Committee.**		*Delete as appropriate*
3.1	Does your research involve participants who are under the age of 18?	**NO**
3.2	Does your research involve adults who are vulnerable because of their social, psychological or medical circumstances (vulnerable adults)? *This includes adults with cognitive and / or learning disabilities, adults with physical disabilities and older people.*	**NO**
3.3	Are participants recruited because they are staff or students of City, University of London? *For example, students studying on a particular course or module.* *If yes, then approval is also required from the Head of Department or Programme Director.*	**NO**
3.4	Does your research involve intentional deception of participants?	**NO**
3.5	Does your research involve participants taking part without their informed consent?	**NO**
3.5	Is the risk posed to participants greater than that in normal working life?	**NO**
3.7	Is the risk posed to you, the researcher(s), greater than that in normal working life?	**NO**

A.4 If you answer YES to the following question and your answers to all other questions in sections A1, A2 and A3 are NO, then your project is deemed to be of MINIMAL RISK. **If this is the case, then you can apply for approval through your supervisor under PROPORTIONATE REVIEW. You do so by completing PART B of this form.** **If you have answered NO to all questions on this form, then your project does not require ethical approval. You should submit and retain this form as evidence of this.**		*Delete as appropriate*
4	Does your project involve human participants or their identifiable personal data? *For example, as interviewees, respondents to a survey or participants in testing.*	**YES**

PART B: Ethics Proportionate Review Form

If you answered YES to question 4 and NO to all other questions in sections A1, A2 and A3 in PART A of this form, then you may use PART B of this form to submit an application for a proportionate ethics review of your project. Your project supervisor has delegated authority to review and approve this application under proportionate review. You must receive final approval from your supervisor in writing before beginning the planned research.

However, if you cannot provide all the required attachments (see B.3) with your project proposal (e.g. because you have not yet written the consent forms, interview schedules etc), the approval from your supervisor will be ***provisional***. You **must** submit the missing items to your supervisor for approval prior to commencing these parts of your project. Once again, you must receive written confirmation from your supervisor that any provisional approval has been superseded by with ***full approval*** of the planned activity as detailed in the full documents. **Failure to follow this procedure and demonstrate that final approval has been achieved may result in you failing the project module.**

Your supervisor may ask you to submit a full ethics application through Research Ethics Online, for instance if they are unable to approve your application, if the level of risks associated with your project change, or if you need an approval letter from the CSREC for an external organisation.

B.1 The following questions must be answered fully. All grey instructions must be removed.		*Delete as appropriate*
1.1.	Will you ensure that participants taking part in your project are fully informed about the purpose of the research?	**YES**
1.2	Will you ensure that participants taking part in your project are fully informed about the procedures affecting them or affecting any information collected about them, including information about how the data will be used, to whom it will be disclosed, and how long it will be kept?	**YES**
1.3	When people agree to participate in your project, will it be made clear to them that they may withdraw (i.e. not participate) at any time without any penalty?	**YES**
1.4	Will consent be obtained from the participants in your project? Consent from participants will be necessary if you plan to involve them in your project or if you plan to use identifiable personal data from existing records. "Identifiable personal data" means data relating to a living person who might be identifiable if the record includes their name, username, student id, DNA, fingerprint, address, etc.	**YES**

	If YES, you must attach drafts of the participant information sheet(s) and consent form(s) that you will use in section B.3 or, in the case of an existing dataset, provide details of how consent has been obtained. *You must also retain the completed forms for subsequent inspection. Failure to provide the completed consent request forms will result in withdrawal of any earlier ethical approval of your project.*	
1.5	Have you made arrangements to ensure that material and/or private information obtained from or about the participating individuals will remain confidential?	**YES**

B.2 If the answer to the following question (B2) is YES, you must provide details		*Delete as appropriate*
2	Will the research be conducted in the participant's home or other non-University location?	**YES**

B.3 Attachments **ALL of the following documents MUST be provided to supervisors if applicable.** **All must be considered prior to final approval by supervisors.** **A written record of final approval must be provided and retained.**	***YES***	***NO***	***Not Applicable***
Details on how safety will be assured in any non-University location, including risk assessment if required (see B2)	Surveys and Questionnaires will be sent out via the internet to be answered in any location. Participants will be required to confirm they are over the age of 18 before participating in surveys and questionnaires.		
Details of arrangements to ensure that material and/or private information obtained from or about the participating individuals will remain confidential (see B1.5) *Any personal data must be acquired, stored and made accessible in ways that are GDPR compliant.*	Participants will be required to give their email address to confirm that they are not a bot and therefore are answering reliably. This information will be stored confidentially through Google surveys / forms. And participants can remove their contributions and information at request.		
Full protocol for any workshops or interviews**			
Participant information sheet(s)**			
Consent form(s)**			

Questionnaire(s)** *sharing a Qualtrics survey with your supervisor is recommended.*			
Topic guide(s) for interviews and focus groups**			
Permission from external organisations or Head of Department** *e.g. for recruitment of participants*			

***If these items are not available at the time of submitting your project proposal, then **provisional approval** can still be given, under the condition that you must submit the final versions of all items to your supervisor for approval at a later date. **All** such items **must** be seen and approved by your supervisor before the activity for which they are needed begins. Written evidence of **final approval** of your planned activity must be acquired from your supervisor before you commence.*

Changes

If your plans change and any aspects of your research that are documented in the approval process change as a consequence, then any approval acquired is invalid. If issues addressed in Part A (the checklist) are affected, then you must complete the approval process again and establish the kind of approval that is required. If issues addressed in Part B are affected, then you must forward updated documentation to your supervisor and have received written confirmation of approval of the revised activity before proceeding.

Templates for Consent and Information

You must use the templates provided by the University as the basis for your participant information sheets and consent forms. You **must** adapt them according to the needs of your project before you submit them for consideration.

Participant Information Sheets, Consent Forms and Protocols must be consistent. Please ensure that this is the case prior to seeking approval. Failure to do so will slow down the approval process.

We strongly recommend using Qualtrics to produce digital information sheets and consent forms.

Further Information

http://www.city.ac.uk/department-computer-science/research-ethics

https://www.city.ac.uk/research/ethics/how-to-apply/participant-recruitment

https://www.city.ac.uk/research/ethics

Appendix B: Reuse Summary

In the development of the 2D platformer game for this dissertation, the A* Pathfinding Project by Aron Granberg was employed as a significant resource to enhance the game's AI and navigation system. This section outlines the rationale behind this project's use and describes how it was integrated into the Unity 2D platformer.

The A* Pathfinding Project is a widely acclaimed, robust, and efficient pathfinding solution for Unity, suitable for various game genres. Its incorporation into the 2D platformer game streamlined the development process, allowing for a focus on the innovative mechanics and features unique to the game rather than dedicating substantial time to implementing a custom pathfinding system.

The following summarises the specific aspects of the A* Pathfinding Project that were reused in the 2D platformer game:

- Grid Graph: Utilised to create a navigable grid representing the game's 2D environment, this feature enabled the efficient calculation of optimal paths for AI characters, accounting for obstacles and platform layouts.
- Pathfinding Algorithms: The A* Pathfinding Project provides several algorithms, such as A*, Dijkstra, and Jump Point Search. The A* algorithm was chosen due to its balance between performance and accuracy in determining the shortest path.
- AI Path Component: This component was attached to AI characters, allowing them to follow paths generated by the A* algorithm. It was customised to fit the unique requirements of the 2D platformer game, such as handling vertical movement, wall-jumping, and other platform-specific actions.
- Seeker Component: The Seeker component managed path requests and optimised performance by caching and updating paths as required. Doing so enabled the game to maintain a smooth frame rate, even with multiple AI characters navigating simultaneously.

6,950 lines of code were reused from the A* Pathfinding Project, and its integration allowed the development of an engaging and dynamic AI system while saving time and resources.

Acknowledgement and credit to Aron Granberg's A* Pathfinding Project are provided in the game's documentation and within the source code where appropriate. The project's licensing terms have been duly adhered to, and no proprietary components or assets have been violated.

By utilising the A* Pathfinding Project, the 2D platformer game benefits from a sophisticated and efficient pathfinding system, enhancing the overall gameplay experience and demonstrating the successful integration of third-party resources in game development.

Appendix C: Manual Testing (Test Cases)

ID:	1	Objective:	1.1 Develop Terrain with Collision Detection
Rationale:			
Ensure the terrain has proper collision detection, allowing correct interaction between the player, game objects, and environment.			
Steps:		Results:	
Start a new game. Move the player around the terrain, attempting to walk through walls, platforms, and other objects. Try to jump onto platforms and through walls. Observe the player's interactions with the environment.		The player should not be able to pass through walls or objects. The player should be able to jump onto platforms and land on them correctly. Collision detection should work consistently and predictably.	
Build 1 Test Result:		PASSED	
Build 2 Test Result:		PASSED	
Build 3 Test Result:		PASSED	

ID:	2	Objective:	1.2 Design a Player with Movement Capabilities
Rationale:			
Ensure the player character has smooth and responsive movement mechanics like walking, running, and jumping.			
Steps:		Results:	
Start a new game. Move the player character around the terrain using the movement controls. Attempt to walk, run, and jump with the player character. Observe the player character's animation states as they perform different actions.		The player character should be able to walk, run, and jump responsively. Animation states should change correctly based on player actions. Movement mechanics should feel fluid and enjoyable.	
Build 1 Test Result:		PASSED	
Build 2 Test Result:		PASSED	
Build 3 Test Result:		PASSED	

ID:	3	Objective:	1.3 Develop a Camera System that Tracks the Player
Rationale:			
To ensure the camera follows the player's character, maintaining their position at the centre of the screen for a seamless and engaging gameplay experience.			
Steps:			Results:
Start a new game. Move the player character around the terrain. Observe the camera movement as the player moves.			The camera should follow the player's character, keeping them centred on the screen. Camera movement should be smooth and provide a clear view of the game world.
Build 1 Test Result:			PASSED
Build 2 Test Result:			PASSED
Build 3 Test Result:			PASSED

ID:	4	Objective:	1.4 Integrate Basic Sound Effects for Player Actions
Rationale:			
To ensure sound effects correspond to player actions and improve game immersion.			
Steps:			Results:
Start a new game. Perform various player actions, such as walking, running, jumping, and interacting with objects. Listen for the sound effects triggered by each action.			Sound effects should play corresponding to the player's actions. Sound effects should enhance immersion and provide feedback on player interactions.
Build 1 Test Result:			PASSED
Build 2 Test Result:			PASSED
Build 3 Test Result:			PASSED

ID:	5	Objective:	1.5 Design Obstacles for Player Challenges
Rationale:			
To ensure various obstacles have been designed to present challenges to the player and enhance gameplay depth.			
Steps:		Results:	
Start a new game. Traverse the level, encountering various obstacles. Attempt to overcome the obstacles using the player character's abilities. Observe the functionality of the obstacles and their interaction with the player.		Obstacles should be well-designed and functional. Obstacles should challenge the player, requiring skill and strategy to overcome.	
Build 1 Test Result:		PASSED	
Build 2 Test Result:		PASSED	
Build 3 Test Result:		PASSED	

ID:	6	Objective:	1.6 Develop Enemies with Basic AI
Rationale:			
To ensure simple enemy characters with AI behaviours are present, offering an additional challenge to the player.			
Steps:		Results:	
Start a new game. Encounter enemy characters throughout the level. Observe enemy AI behaviours, such as movement and reactions to the player's actions. Engage in combat with the enemies.		Enemy characters should have basic AI behaviours. AI behaviours should create dynamic and engaging encounters.	
Build 1 Test Result:		PASSED	
Build 2 Test Result:		PASSED	
Build 3 Test Result:		PASSED	

ID:	7	Objective:	1.7 Establish a Player Respawn System
Rationale:			
Ensure a respawn system is in place to maintain game flow and reduce frustration when the player dies.			
Steps:			Results:
Start a new game. Intentionally cause the player character to die. Observe the respawn system in action. Check the starting point after respawning.			The player character should respawn after dying. The player should restart from the beginning of the current level. The respawn system should maintain game flow and reduce frustration.
Build 1 Test Result:			PASSED
Build 2 Test Result:			PASSED
Build 3 Test Result:			PASSED

ID:	8	Objective:	1.8 Set Level Start and Finish Points
Rationale:			
To ensure clearly defined starting and ending points for the level, providing a sense of progression and achievement.			
Steps:			Results:
Start a new game. Identify the starting point of the level. Traverse the level and reach the finish point. Observe the level transition or ending sequence.			The starting point should be distinct and clear. The finish point should be discrete and invisible. Upon reaching the finish point, the player has a sense of progression and achievement.
Build 1 Test Result:			PASSED
Build 2 Test Result:			PASSED
Build 3 Test Result:			PASSED

ID:	9	Objective:	1.9 Develop Bullet Collection and Switching Mechanism
Rationale:			
To ensure players can collect and switch between various bullet types, allowing them to adapt their strategy for different situations and enemy encounters.			
Steps:			Results:
Start a new game. Collect different bullet types throughout the level. Switch between collected bullet types using the designated controls. Engage in combat with enemies using different bullet types.			Players should be able to collect various bullet types. Players should be able to switch between collected bullet types seamlessly. Different bullet types should have noticeable effects on gameplay and enemy encounters.
Build 1 Test Result:			PASSED
Build 2 Test Result:			PASSED
Build 3 Test Result:			PASSED

ID:	10	Objective:	1.10 Develop the Main Character's Invisibility feature
Rationale:			
Ensure the invisibility feature works as intended and provides a unique gameplay experience.			
Steps:			Results:
Start a new game. Activate the player character's invisibility feature using the designated controls. Observe the visual effects and the player's vision while invisible. Engage with enemies and obstacles while using the invisibility feature. Attempt to strategise and overcome challenges using the invisibility feature.			The player character should be able to activate the invisibility feature. Visual effects and the player's vision should be altered during invisibility. The invisibility feature should have a noticeable impact on gameplay and enemy behaviour. Players should be able to strategise and overcome challenges using the invisibility feature.
Build 1 Test Result:			PASSED
Build 2 Test Result:			PASSED
Build 3 Test Result:			PASSED

ID:	11	Objective:	2.1 Design a Player Tutorial
Rationale:			
To introduce players to the game's essential mechanics and features, ensuring they can effectively engage with the game's features and controls.			
Steps:		Results:	
Start a new game. Access the tutorial level from the start menu. Complete the tutorial, learning the game's essential mechanics and features. Evaluate the effectiveness of the tutorial in teaching the player about the game's controls, invisibility feature, and bullet collection and switching system.		Players should be able to access the tutorial level from the start menu. The tutorial should effectively teach the game's essential mechanics and features. The player should feel comfortable with the controls and game mechanics upon completion of the tutorial. The tutorial should be user-friendly and provide a clear understanding of the game's features and controls.	
Build 2 Test Result:		SEMI PASS – NEEDS UI	
Build 3 Test Result:		PASSED	

ID:	12	Objective:	2.2 Develop a Start Menu Interface
Rationale:			
To provide players with a polished and professional game experience, allowing them to start the game, access the tutorial, and view credits.			
Steps:		Results:	
Launch the game. Evaluate the start menu interface for design and functionality. Access the tutorial, start the game, and view credits from the start menu.		The start menu should be well-designed and visually appealing. The start menu should be easy to navigate and provide access to the game, tutorial, and credits. All options in the start menu should function as intended.	
Build 2 Test Result:		PASSED	
Build 3 Test Result:		PASSED	

ID:	13	Objective:	2.3 Create a Second Level with Boss Fight
Rationale:			
To provide players with a challenging and memorable gameplay experience that tests their platforming and invisibility use skills.			

Steps:	Results:
Start a new game. Complete the first level and progress to the second level. Traverse the second level, using platforming and invisibility skills to overcome obstacles quickly enough to evade the boss enemy. Evaluate the level design, challenge, and enjoyment of the second level.	The second level should be well-designed and engaging. The boss fight should present a significant challenge, testing the player's platforming and invisibility use skills. The second level should provide a sense of accomplishment and progression upon completion.
Build 2 Test Result:	PASSED
Build 3 Test Result:	PASSED

ID:	14	Objective:	2.4 Enhance Enemy AI
Rationale:			
To improve the game's overall challenge and engagement through dynamic enemy behaviour based on the player's visibility status.			

Steps:	Results:
Start a new game. Encounter various enemy types throughout the game. Observe enemy behaviour when the player is visible and invisible. Evaluate the enemy AI's effectiveness in creating diverse and exciting gameplay scenarios.	Enemies should exhibit dynamic behaviour based on the player's visibility status. The enhanced enemy AI should provide a more challenging and engaging gameplay experience. The invisibility feature should be effectively utilised in response to the improved enemy AI.
Build 2 Test Result:	SEMI PASS – NEEDS FINAL BOSS
Build 3 Test Result:	PASSED

ID:	15	Objective:	2.5 Add Sound Effects for the Enemies and Environment
Rationale:			
To enhance the game's atmosphere and immersion and provide important gameplay feedback.			
Steps:			Results:
Start a new game. Listen for the enemy and environmental sound effects while playing. Assess the quality and effectiveness of the sound effects in creating an immersive experience and providing gameplay feedback.			Sound effects should be present for enemy movement, attacks, and death animations. Environmental sounds should create a sense of realism and depth in the game world. The sound effects should contribute to a more engaging and enjoyable player experience by providing important gameplay feedback and enhancing the game's atmosphere.
Build 2 Test Result:			PASSED
Build 3 Test Result:			PASSED

ID:	16	Objective:	2.6 Implement a Timer Score System
Rationale:			
To encourage players to improve their skills, strive for faster completion times, and allow for friendly competition among players.			
Steps:			Results:
Start a new game. Complete each level while the timer tracks progress. Observe the timer score system in action, noting completion times for each level. Evaluate the effectiveness of the timer score system in adding challenge, replayability, and competition to the game.			The timer score system should accurately track the player's progress through each level. Players should be encouraged to complete levels more quickly and efficiently. The timer score system should provide the game with challenge and replayability.
Build 2 Test Result:			PASSED
Build 3 Test Result:			PASSED

ID:	17	Objective:	2.7 Implement a UI for the Bullet System
Rationale:			
To provide players with clear, easily accessible information about their combat resources, facilitating strategic choices and enhancing overall player agency.			
Steps:			Results:
Start a new game. Observe the bullet system UI during gameplay. Switch between bullet types and monitor remaining ammunition using the bullet system UI. Evaluate the UI for visual clarity, ease of use, and efficient information communication.			The bullet system UI should display the player's current bullet type, remaining ammunition, and available bullet types. The UI should be visually clear and easy to use during gameplay. The bullet system UI should contribute to a seamless gameplay experience by minimising the need for pausing or navigating complex menus.
Build 2 Test Result:			PASSED
Build 3 Test Result:			PASSED

ID:	18	Objective:	3.1 Create a Third Level with a Double Jump Ability
Rationale:			
To provide players with increased mobility and expand the game's platforming mechanics, allowing them to reach higher platforms and avoid more dangerous obstacles.			
Steps:			Results:
Play the third level, focusing on the double jump ability. Test the double jump ability in various scenarios, including reaching higher platforms and avoiding dangerous obstacles. Verify that the double jump ability enhances the player's mobility and expands the game's platforming mechanics. Check for any bugs or issues related to the double jump ability. Evaluate the overall gameplay experience with the double jump ability integrated.			The double jump ability should function correctly and consistently. The double jump ability should enhance the player's mobility and expand the game's platforming mechanics. The player should be able to use the double jump ability to reach higher platforms and avoid dangerous obstacles. No significant bugs or issues related to the double jump ability should be present. The gameplay experience should be smooth and enjoyable with the integrated double jump ability.
Build 3 Test Result:			PASSED

ID:	19	Objective:	3.2. Implement More Challenging Obstacles for the Third Level
Rationale:			
Ensure that the third level provides a more challenging and engaging gameplay experience.			
Steps:		Results:	
Play the third level, focusing on the new challenging obstacles, such as invisible blades and spikes. Test players' ability to navigate and overcome these obstacles using their platforming skills and double jump ability. Assess the difficulty and balance of the new obstacles in relation to the player's skills and abilities. Check for any issues or bugs related to the new obstacles. Evaluate the overall gameplay experience with the new obstacles integrated into the third level.		The new challenging obstacles should function correctly and consistently. The player should be able to navigate and overcome the new obstacles using their platforming skills and double jump ability. The new obstacles should provide a balanced and challenging gameplay experience. No significant bugs or issues should be present that are related to the new obstacles. The overall gameplay experience should be engaging and enjoyable with the new obstacles integrated into the third level.	
Build 3 Test Result:		PASSED	

ID:	20	Objective:	3.3. Create a More Challenging Boss Enemy for Level 4 (The Final Level)
Rationale:			
To ensure that level 4 provides a memorable and engaging gameplay experience, challenging the player with a difficult final boss enemy.			
Steps:		Results:	
Play level 4, focusing on the more challenging boss enemy. Test the player's ability to defeat the boss using their platforming skills, invisibility feature, and combat elements. Assess the difficulty and balance of the boss fight in relation to the player's skills and abilities. Check for any issues or bugs related to the more challenging boss enemy. Evaluate the overall gameplay experience with the more challenging boss enemy integrated into level 4.		The more challenging boss enemy should function correctly and consistently. The player should be able to defeat the boss using their platforming skills, invisibility feature, and combat elements. The boss fight should provide a balanced and challenging gameplay experience. No significant bugs or issues related to the more challenging boss enemy should be present. The overall gameplay experience should be memorable and engaging with the more challenging boss enemy integrated into level 4.	
Build 3 Test Result:		PASSED	

ID:	21	Objective:	3.4. Implement an End Screen That Shows the Final Time of the Player's Game Run
Rationale:			
The end screen should correctly display the player's final time and provide a visually appealing and informative presentation of this scoring element.			
Steps:			Results:
Complete the game, reaching the end screen. Verify that the end screen displays the player's final time accurately. Check the end screen's visual design, ensuring it is consistent with the game's overall style and aesthetics. Assess the clarity and effectiveness of the end screen's presentation, including the time display and any animations. Evaluate the overall player experience with the end screen, considering satisfaction, competitiveness, and replayability factors.			The end screen should accurately display the player's final time. The end screen's visual design should be consistent with the game's overall style and aesthetics. The end screen's presentation should be clear and effective, prominently displaying the player's final time and incorporating relevant animations. The end screen should contribute positively to the overall player experience, encouraging satisfaction, competitiveness, and replayability. No significant bugs or issues should be present related to the end screen.
Build 3 Test Result:			PASSED

ID:	22	Objective:	3.5. Add Music to the Game
Rationale:			
The background music should effectively enhance the game's atmosphere and overall player experience.			
Steps:			Results:
Play through each level, menu, and victory screen with the background music enabled. Assess the quality and appropriateness of the music for each level, menu, and victory screen. Verify that the music contributes positively to the game's atmosphere and player experience. Check for any issues or bugs related to the integration of music into the game. Evaluate the overall impact of the music on the game's aesthetics, mood, and enjoyment.			The background music should be of high quality and appropriate for each level, menu, and victory screen. The music should contribute positively to the game's atmosphere and overall player experience. No significant issues or bugs should be present related to the integration of music into the game. The overall impact of the music on the game's aesthetics, mood, and enjoyment should be positive and engaging.
Build 3 Test Result:			PASSED

ID:	23	Objective:	3.6. Polish and Fine Tune the Game
Rationale:			
The game needs to be polished, free of major bugs, and provide a satisfying and enjoyable experience for the player.			

Steps:	Results:
Play through the entire game, paying close attention to visual and audio assets, gameplay elements, and overall polish. Identify any remaining bugs, issues, or inconsistencies and document them for further refinement. Assess gameplay elements' balance and overall quality, including platforming, combat, and level design. Verify that the visual and audio assets are polished and cohesive, contributing to a high-quality final product. Evaluate the overall satisfaction and enjoyment of the gameplay experience, considering factors such as player agency, challenge, and replayability.	The game should be polished and free of major bugs or issues. The visual and audio assets should be of high quality and contribute to a cohesive and engaging game world. The gameplay elements should be well-balanced and enjoyable, providing a satisfying and challenging experience for the player. The overall satisfaction and enjoyment of the gameplay experience should be high, considering factors such as player agency, challenge, and replayability. The final product should be of high quality, providing players with a satisfying and enjoyable experience.
Build 3 Test Result:	PASSED

Appendix D: Complete Records of User Testing Consent Form, Information Sheet, Questionnaires, and Interviews

Participant Consent Form Template

Name of Principal Researcher: William Dibble

Title of Study: Researching the effectiveness of an Invisibility feature within a 2D Platformer video game that adapts the enemy AI based on this feature.

Participant Number:

Please tick or

initial box

1	I confirm that I have read and understood the participant information dated 20/03/2023 for the above study. I have had the opportunity to consider the information and ask questions which have been answered satisfactorily.	
2.	I understand that my participation is voluntary and that I am free to withdraw without giving a reason without being penalised or disadvantaged.	
3.	I agree to the interview being audio OR video recorded.	
4.	I agree to City recording and processing this information about me. I understand that this information will be used only for the purpose(s) explained in the participant information and my consent is conditional on City complying with its duties and obligations under the General Data Protection Regulation (GDPR).	
5.	I agree to take part in the above study.	
6.	I understand that my feedback within interviews and/or questionnaires will be used within the final project submission (e.g. direct quotes).	

____________________ ____________________________ _____________

Name of Participant	Signature	Date
______________________	_______________________________	______________
Name of Researcher	Signature	Date

Participant User Testing Information Sheet

Date: 2023-03-21

Version: 1.0

Title of study: Researching the effectiveness of an Invisibility feature within a 2D Platformer video game that adapts the enemy AI based on this feature.

Name of principal investigator/researcher: William Dibble

We would like to invite you to take part in a research study focused on the development of a 2D platformer game for a BSc Computer Science project. Before you decide whether you would like to take part, it is important that you understand why the research is being done and what it would involve for you. Please take time to read the following information carefully and ask us if there is anything that is not clear, or if you would like more information. You will be given a copy of this information sheet to keep.

What is the purpose of the study?

The aim of this study is to create a functional and enjoyable 2D platformer game by incorporating participant feedback. The study will take place over a period of 1 month.

Why have I been invited to take part?

You have been invited to take part in this study because you have experience in playing video games. We are seeking participants with varying levels of gaming experience, from beginners to advanced players. Approximately 8 participants will be involved in the study.

Do I have to take part?

Participation in the project is voluntary, and you can choose not to participate in part or all of the project. You can withdraw at any stage of the project without being penalised or disadvantaged in any way. It is up to you to decide whether or not to take part. If you do decide to take part, you will be asked to sign a consent form. If you decide to take part, you are still free to withdraw at any time and without giving a reason.

What will happen if I take part?

If you agree to take part, you will be asked to:

- Playtest the 2D platformer game at various stages of development (total time commitment: 1-2 hours)
- Provide feedback on the game mechanics, level design, user interface, and overall enjoyment by completing qualitative questionnaires (5-10 minutes)
- Engage in short semi-structured interviews to share your experiences with the game (10-15 minutes)

The research will take place at the researchers place of residency: 18 Fulford Road, Epsom, KT199QX.

What instructions will I need to follow when playtesting?

When playtesting, the researcher will run the session as follows:

- Introduction to the game: Briefly introduce the game, its genre, and the main objectives to the participant.
- Explain the controls: Provide the participant with a clear overview of the game controls, including movement, jumping, shooting, and switching between bullet types.
- Gameplay session: Allow the participant to play the game for a set amount of time (e.g., 30 minutes) or until they complete the demo, whichever comes first.
- Encourage open feedback: Encourage the participant to share their thoughts and feedback on the game during the playtesting session, including any bugs or issues they encounter.
- Note-taking: Take notes on the participant's playstyle, areas where they struggle, and any comments they make during the playtesting session.

How will the post playtesting interviews be structured?

The interviews will be semi structured and consist of set and improvised questions, they should be approximately 10 minutes long. The first four semi-structured interview questions will be as follows:

- What are your overall impressions of the game? Did you find it enjoyable to play?
- How would you rate the difficulty of the game? Did you find it too easy, too hard, or just right?
- What are your thoughts on the invisibility feature? How did it impact your gameplay experience?
- How did you find the enemy AI behaviour? Did it adapt well to the invisibility feature?

Based on the participant's answers, follow-up questions will be asked to gather more detailed information on their experience:

- (If the participant found the game too easy or too hard) Can you provide specific examples of parts of the game that you found too easy or too hard? What changes would you suggest to improve the difficulty balance?
- (If the participant had issues with the controls) Were there any controls that you found particularly challenging or unintuitive? How would you suggest improving them?
- (If the participant had suggestions for the invisibility feature) How would you change the invisibility feature to make it more engaging or better integrated into the gameplay experience?
- (If the participant encountered any bugs or issues) Can you describe the bugs or issues you encountered during the playtesting session? How did they affect your experience with the game?

Finally, open-ended questions will be asked to gather additional feedback and insights such as:

- What aspects of the game's level design, user interface, or game mechanics stood out to you as particularly effective or problematic?
- Are there any additional features or improvements you could suggest that would the overall gameplay experience?

What are the possible disadvantages and risks of taking part?

There are no anticipated risks or discomforts associated with participating in this study. However, if you experience any discomfort or stress while playtesting, you may take breaks or discontinue participation at any time.

What are the possible benefits of taking part?

By participating in this study, you will contribute to the improvement of the game's design, functionality, and overall experience. Your feedback will help identify potential issues and areas for enhancement, ultimately leading to a better final product.

Expenses and Payments:

No expenses or payments will be offered to participants in this study.

How is the project being funded?

The project is self-funded by the researcher as part of their BSc Computer Science degree.

Conflicts of interests:

There are no known conflicts of interest associated with this research project.

What should I do if I want to take part?

If you are interested in participating, please contact the researcher, William Dibble, at William.dibble@city.ac.uk.

Data privacy statement:

City, University of London is the sponsor and the data controller of this study based in the United Kingdom. This means that we are responsible for looking after your information and using it properly. The legal basis under which your data will be processed is City's public task.

Your right to access, change or move your information are limited, as we need to manage your information in a specific way in order for the research to be reliable and accurate. To safeguard your rights, we will use the minimum personal-identifiable information possible (for further information please see https://ico.org.uk/for-organisations/guide-to-data-protection/guide-to-the-general-data-protection-regulation-gdpr/lawful-basis-for-processing/public-task/).

City will use your name and contact details to contact you about the research study as necessary. The only person at City who will have access to your identifiable information will be William Dibble. City will keep identifiable information about you from this study for 1 year after the study has finished.

You can find out more about how City handles data by visiting https://www.city.ac.uk/about/governance/legal. If you are concerned about how we have processed your personal data, you can contact the Information Commissioner's Office (IOC) https://ico.org.uk/.

Will my taking part in the study be kept confidential?

Your confidentiality will be maintained throughout the study. The following measures are in place to ensure this:

- Personal data access: Only the researcher will have access to your personal data before it is anonymised.
- Ensuring confidentiality/anonymity: Identifiable information will not be recorded, and any quotes used will be anonymised or paraphrased to protect your identity.
- Audio/video recording/photographs: Any recordings or photographs taken during the study will be treated with the same confidentiality measures.
- Future data use: If there is a request for future use of data, such as contacting participants for another study, you will be informed, and your explicit consent will be required.
- Confidentiality restrictions: Please note that confidentiality may be limited in cases where reporting of violence, abuse, self-inflicted harm, harm to others, or criminal activity is necessary.
- Data storage and destruction: Records will be securely stored (e.g., on OneDrive, in a locked filing cabinet, or on an encrypted laptop) for a period of 10 years, as per City University's guidelines, and will be destroyed after that period.
- Publishing data: Data may be published via the UK Data Archive or Figshare in line with Open Scholarship principles, ensuring that anonymity is maintained.

What will happen to the results?

The results may be used for various publications, including the current thesis/report and potential future publications. Anonymity will be maintained in all cases. If you wish to receive a copy of the publication or a summary of the results, please provide your contact details for this purpose, ensuring you explicitly consent to your data being retained for this reason.

Who has reviewed the study?

This study has been approved by the City, University of London Computer Science Research Ethics Committee (CSREC).

What if there is a problem?

If you have any problems, concerns, or questions about this study, you should ask to speak to a member of the research team. If you remain unhappy and wish to complain formally, you can do this through City's complaints procedure. To complain about the study, you need to phone 020 7040 3040. You can then ask to speak to the Secretary to Senate Research Ethics Committee and inform them that the name of the project is *"Researching the effectiveness of an Invisibility feature within a 2D Platformer video game that adapts the enemy AI based on this feature"*.

You can also write to the Secretary at:

Annah Whyton

Research & Enterprise Office

City, University of London

Northampton Square

London, EC1V 0HB

Email: senaterec@city.ac.uk

Further information and contact details:

William Dibble – william.dibble@city.ac.uk

Thank you for taking the time to read this information sheet.

Post-Playtesting Questionnaire Template

Question 1: How experienced are you with playing video games?

- Beginner
- Intermediate
- Advanced

Question 2: How familiar are you with 2D platformer games?

- Not familiar at all
- Somewhat familiar
- Very Familiar

Question 3: How would you rate the overall game design and visuals?

- Poor
- Fair
- Good
- Excellent

Question 4: How did you find the game's difficulty level?

- Too easy
- Just right
- Too hard

Question 5: On a scale of 1 to 4, how well do you agree with this statement? "The game controls were intuitive and easy to use".

- 1 (Strongly disagree)
- 2
- 3
- 4 (Strongly agree)

Question 6: On a scale of 1 to 4, do you feel that the invisibility feature was necessary to complete the game levels?

- 1 (Not at all)
- 2
- 3
- 4 (Yes, significantly)

Question 7: On a scale of 1 to 4, did the uniqueness of the invisibility feature make the game more enjoyable?

- 1 (Strongly disagree)
- 2
- 3
- 4 (Strongly agree)

Question 8: Did you encounter any issues or problems while playing the game?

(Short answer)

Question 9: On a scale of 1 to 4, overall, did you enjoy playing the game?

- 1 (Not at all)
- 2

- 3
- 4 (Yes, significantly)

Participant 1

Questionnaire Answers for Participant 1

Question 1: How experienced are you with playing video games?

- Intermediate

Question 2: How familiar are you with 2D platformer games?

- Very Familiar

Question 3: How would you rate the overall game design and visuals?

- Good

Question 4: How did you find the game's difficulty level?

- Just right

Question 5: On a scale of 1 to 4, how well do you agree with this statement? "The game controls were intuitive and easy to use".

- 3

Question 6: On a scale of 1 to 4, do you feel that the invisibility feature was necessary to complete the game levels?

- 4 (Yes, significantly)

Question 7: On a scale of 1 to 4, did the uniqueness of the invisibility feature make the game more enjoyable?

- 4 (Strongly agree)

Question 8: Did you encounter any issues or problems while playing the game?

- The text in some parts of the game was a bit small and difficult to read. Switching between bullet types took a bit long.

Question 9: On a scale of 1 to 4, overall, did you enjoy playing the game?

- 4 (Yes, significantly)

Participant 1 Interview Transcript:

Interviewer: What are your overall impressions of the game? Did you find it enjoyable to play?

Participant 1: I actually found the game quite enjoyable. The graphics and level design were good, and the controls felt, um, smooth and responsive. I liked the challenge the game had, and I think the invisibility feature added an interesting twist.

Interviewer: How would you rate the difficulty of the game? Did you find it too easy, too hard, or just right?

Participant 1: Hm, I guess I'd say the difficulty was just right. It wasn't, like, too easy, so I felt accomplished when I completed levels, but it also wasn't too hard that it became annoying, you know? I think the difficulty curve throughout the game was balanced well.

Interviewer: What are your thoughts on the invisibility feature? How did it impact your gameplay experience?

Participant 1: Yeah, I think it added, like, an extra layer of strategy to the game, so it was more than just a normal platformer. I was planning my moves more carefully and timing my invisibility to avoid the enemies. It made the game more engaging for me.

Interviewer: How did you find the enemy AI behaviour? Did it adapt well to the invisibility feature?

Participant 1: I think the enemy AI was well done. It responded well to the invisibility feature and seemed to adapt its behaviour when I was invisible. I noticed that enemies were like searching for me when I was invisible, and they changed how they moved based on if they could see me.

Interviewer: Can you provide specific examples of parts of the game that you found particularly engaging or challenging?

Participant 1: Um, there was the level near the beginning where I had to get through the platforms underground with multiple of those jumping enemies around and with the invisibility, I could dodge them. It felt good when I managed to complete that level.

Interviewer: Were there any controls that you found particularly challenging or unintuitive? How would you suggest improving them?

Participant 1: Well, the controls were pretty fine and easy to use. But switching between bullet types was a bit, like, long. Maybe assigning different bullet types to different keys would make it easier to switch between them quickly?

Interviewer: What aspects of the game's level design, user interface, or game mechanics stood out to you as particularly effective or problematic?

Participant 1: Level design seemed well thought out. The levels offered a good mix of platforming and enemy fights, which kept the gameplay interesting. The user interface was pretty straightforward and easy to get. One thing I did notice was that some of the text was a bit small and difficult to read, but was mostly fine.

Interviewer: Are there any additional features or improvements you could suggest that would enhance the overall gameplay experience?

Participant 1: I think maybe add some more variety in terms of enemy types or terrains. I think this would keep the gameplay more interesting throughout it. And maybe having some sort of power-up or upgrade system for the player character could add like depth and make the game even more engaging. Other than that, I think it's a really solid game, and I enjoyed playing it.

Participant 2

Questionnaire Answers for Participant 2

Question 1: How experienced are you with playing video games?

- Beginner

Question 2: How familiar are you with 2D platformer games?

- Not familiar at all

Question 3: How would you rate the overall game design and visuals?

- Good

Question 4: How did you find the game's difficulty level?

- Just right

Question 5: On a scale of 1 to 4, how well do you agree with this statement? "The game controls were intuitive and easy to use".

- 2

Question 6: On a scale of 1 to 4, do you feel that the invisibility feature was necessary to complete the game levels?

- 3

Question 7: On a scale of 1 to 4, did the uniqueness of the invisibility feature make the game more enjoyable?

- 3

Question 8: Did you encounter any issues or problems while playing the game?

- Difficulty distinguishing between the character's visible and invisible states, trouble understanding the shooting controls.

Question 9: On a scale of 1 to 4, overall, did you enjoy playing the game?

- 3

Participant 2 Interview Transcript:

Interviewer: What are your overall impressions of the game? Did you find it enjoyable to play?

Participant 2: I'm not really a gamer, but I did have fun playing. The graphics were nice, and I liked the idea of adding the invisible feature. It took me a while to get a hang of the controls, but once I did, it was alright.

Interviewer: How would you rate the difficulty of the game? Did you find it too easy, too hard, or just right?

Participant 2: At first, I found the game a bit hard because I don't play games really. But I got used to the controls, and I felt the difficulty was okay. It was challenging to me and slightly frustrating, but I think this game is made really for people who play games a lot.

Interviewer: What are your thoughts on the invisibility feature? How did it impact your gameplay experience?

Participant 2: I think it's really interesting. It added an extra thing to the game and made me think more when to use it or not. And it was helpful when I was trying to avoid some of the enemies like the jumping one.

Interviewer: How did you find the enemy AI behaviour? Did it adapt well to the invisibility feature?

Participant 2: They seemed to like, react to the feature, which made them less difficult to avoid when I was invisible. But it was still hard to see where I was going when I had invisibility turned on, so it wasn't much easier, just different.

Interviewer: Were there any controls that you found particularly challenging or unintuitive? How would you suggest improving them?

Participant 2: I had some trouble getting used to shooting controls so it might be helpful for new players like me if there was more info on the tutorial part to say like how the bullet system works.

Interviewer: How would you change the invisibility feature to make it more engaging or better integrated into the gameplay experience?

Participant 2: I think it would be cool if there were more ways to use it, like maybe being able to interact with certain objects or access hidden areas when you're invisible. That could make it even more fun and make players use it more creatively.

Interviewer: Can you describe any parts of the game that you found particularly enjoyable or memorable?

Participant 2: I thought the big jumping boss part was really hard to play because I'm not good with using the keyboard, but after a while, I got better and felt like I was progressing.

Interviewer: What aspects of the game's level design, user interface, or game mechanics stood out to you as particularly effective or problematic?

Participant 2: I liked the general level design, and the user interface was simple enough for me to understand. Something I had a problem with was that sometimes it was hard to tell when my character was invisible or not. Maybe having a like an outline or something would be helpful to see the player more.

Interviewer: Are there any additional features or improvements you could suggest that would enhance the overall gameplay experience?

Participant 2: I think it would be nice to have some more variety in the types of enemies and obstacles in the game. But other than that, I thought it was really cool.

Participant 3

Questionnaire Answers for Participant 3

Question 1: How experienced are you with playing video games?

- Intermediate

Question 2: How familiar are you with 2D platformer games?

- Somewhat familiar

Question 3: How would you rate the overall game design and visuals?

- Good

Question 4: How did you find the game's difficulty level?

- Just right

Question 5: On a scale of 1 to 4, how well do you agree with this statement? "The game controls were intuitive and easy to use".

- 3

Question 6: On a scale of 1 to 4, do you feel that the invisibility feature was necessary to complete the game levels?

- 3

Question 7: On a scale of 1 to 4, did the uniqueness of the invisibility feature make the game more enjoyable?

- 4 (Strongly agree)

Question 8: Did you encounter any issues or problems while playing the game?

- Difficulty switching between bullet types quickly.

Question 9: On a scale of 1 to 4, overall, did you enjoy playing the game?

- 4 (Yes, significantly)

Participant 3 Interview Transcript:

Interviewer: What are your overall impressions of the game? Did you find it enjoyable to play?

Participant 3: I think the game is quite enjoyable, and it definitely kept me engaged. The concept is interesting, and I appreciate the feature of making the player invisible.

Interviewer: How would you rate the difficulty of the game? Did you find it too easy, too hard, or just right?

Participant 3: I'd say the difficulty was just right for me. There were some difficult things, but nothing that made me feel too annoyed by it.

Interviewer: What are your thoughts on the invisibility feature? How did it impact your gameplay experience?

Participant 3: I really liked the invisibility. It added an extra layer of planning your moves to the game and made it more interesting. It was fun to use it to make it so the bad guys couldn't see me anymore.

Interviewer: How did you find the enemy AI behaviour? Did it adapt well to the invisibility feature?

Participant 3: I thought the enemies moving and stuff seemed pretty good. It looked like they would change when I used the invisibility and became more, like, cautious, which made the game more fun to play through.

Interviewer: You mentioned that you enjoyed the strategic aspect of the invisibility feature. Can you think of any moments in the game where you had to use the feature in a particularly clever way?

Participant 3: There was one bit where I had to get through a bunch of enemies with different movements. I had to time when I used the invisibility feature so I could get past them without being seen. It was rewarding when I got to the end.

Interviewer: As someone with some past experience in gaming, did you find the controls easy to pick up, or were there any aspects that you found difficult or unintuitive?

Participant 3: For the most part, the controls were easy. The only aspect I found a bit unintuitive was switching between bullet types. I eventually got used to it, but it took a little longer than the other controls, especially when you needed to switch it quickly.

Interviewer: Based on your experience with the enemy AI, are there any improvements or additional behaviours you'd like to see implemented to make the game even more engaging?

Participant 3: I think it would be cool to have enemies with different powers that make the player change their game plan. Like an enemy that can see the player even when invisible could add an extra layer of challenge.

Interviewer: You mentioned that the difficulty was just right for you. Were there any specific parts of the game that you found more challenging or easier than others? How would you describe the overall difficulty curve?

Participant 3: The difficulty was quite well-balanced. I think the initial levels were sort of easy after getting used to how to play, which meant I was comfortable with the controls. But as the game progressed, the levels became harder, and I had to rely more on the invisibility thing. There were a few difficult parts that took a while to get through, but they were reasonably fine after a while of playing.

Participant 4

Questionnaire Answers for Participant 4

Question 1: How experienced are you with playing video games?

- Advanced

Question 2: How familiar are you with 2D platformer games?

- Very Familiar

Question 3: How would you rate the overall game design and visuals?

- Good

Question 4: How did you find the game's difficulty level?

- Hard

Question 5: On a scale of 1 to 4, how well do you agree with this statement? "The game controls were intuitive and easy to use".

- 3

Question 6: On a scale of 1 to 4, do you feel that the invisibility feature was necessary to complete the game levels?

- 4 (Yes, significantly)

Question 7: On a scale of 1 to 4, did the uniqueness of the invisibility feature make the game more enjoyable?

- 4 (Strongly agree)

Question 8: Did you encounter any issues or problems while playing the game?

- Enemy AI could be improved, particularly in how they react to invisibility.

Question 9: On a scale of 1 to 4, overall, did you enjoy playing the game?

- 4 (Yes, significantly)

Participant 4 Interview Transcript:

Interviewer: What are your overall impressions of the game? Did you find it enjoyable to play?

Participant 4: I found the game enjoyable, and as someone with a lot of experience on games, it was cool to see a 2D platformer with a special mechanic like the invisibility.

Interviewer: How would you rate the difficulty of the game? Did you find it too easy, too hard, or just right?

Participant 4: The game was hard, especially in the second level. But it made it more fun when you got to the end.

Interviewer: What are your thoughts on the invisibility feature? How did it impact your gameplay experience?

Participant 4: I think it added a new level of strategy, really, which I liked. It meant for different approaches to completing the levels, which kept things interesting.

Interviewer: How did you find the enemy AI behaviour? Did it adapt well to the invisibility feature?

Participant 4: The enemy AI was decent but could use some improving I think because sometimes they didn't react as cleverly as I would have expected when I was using invisibility.

Interviewer: You mentioned that the game was somewhat easy, especially in the earlier levels. Can you provide specific examples of parts of the game that you found too easy or lacking in challenge?

Participant 4: In the earlier level, the enemy movements were quite simple and predictable, which made it easy to use the invisibility feature to avoid them. There was also less worry about how long it was taking because there was no monster chasing you like in the second level.

Interviewer: As an experienced gamer, what changes would you suggest to improve the difficulty and make the game more engaging for players like yourself?

Participant 4: I would probably add more variety and complexity to the enemy movement and how they, like, patrol and have more types of obstacles throughout the game. I think having a difficulty setting could allow players to choose a level of challenge that suits their experience in games.

Interviewer: Based on your observation of the enemy AI, what specific improvements would you like to see in their behaviour, especially in relation to the invisibility feature?

Participant 4: I'd like to see the enemy AI have more adaptive and unpredictable behaviour when the player goes invisible. Like they could become more cautious or change their patrol patterns to cover more area, making it more challenging for the player to get past them.

Interviewer: Since you've played many games in the past, are there any other features or mechanics from those games that you think could be incorporated into this game to enhance the overall experience?

Participant 4: I think adding some powers for the player to pick up and use could make the game more interesting. Something like a speed boost or a shield could add more depth to everything and make it more interesting to play.

Participant 5

Questionnaire Answers for Participant 5

Question 1: How experienced are you with playing video games?

- Beginner

Question 2: How familiar are you with 2D platformer games?

- Not familiar at all

Question 3: How would you rate the overall game design and visuals?

- Good

Question 4: How did you find the game's difficulty level?

- Too hard

Question 5: On a scale of 1 to 4, how well do you agree with this statement? "The game controls were intuitive and easy to use".

- 2

Question 6: On a scale of 1 to 4, do you feel that the invisibility feature was necessary to complete the game levels?

- 2

Question 7: On a scale of 1 to 4, did the uniqueness of the invisibility feature make the game more enjoyable?

- 3

Question 8: Did you encounter any issues or problems while playing the game?

- Difficulty in escaping the boss in the second level, initially getting used to controls, and confusion about different bullet types.

Question 9: On a scale of 1 to 4, overall, did you enjoy playing the game?

- 3

Participant 5 Interview Transcript:

Interviewer: What are your overall impressions of the game? Did you find it enjoyable to play?

Participant 5: Yes, it was all quite interesting, considering I don't have much experience with video games. I didn't do very well, but it was still fun to play through something new.

Interviewer: How would you rate the difficulty of the game? Did you find it too easy, too hard, or just right?

Participant 5: For me, I found it very difficult. I have hardly any gaming experience, but I appreciated that the game was approachable, just maybe a bit too tough for me.

Interviewer: What are your thoughts on the invisibility feature? How did it impact your gameplay experience?

Participant 5: It was a fun thing to have, and the animation looks really cool when you do it, but not being able to see the player made things even harder for me, so I didn't really use it much.

Interviewer: How did you find the enemy AI behaviour? Did it adapt well to the invisibility feature?

Participant 5: They did seem to change how they were moving and behaving, but I did have most my attention on not dying every few seconds.

Interviewer: As a newcomer to video games, were there any aspects of the game that you found particularly challenging or confusing?

Participant 5: Initially, getting used to the controls was a bit challenging for me, but after some time, I became more comfortable. The concept of having different bullets was a bit confusing at first, but I got the hang of it.

Interviewer: Did you encounter any obstacles or challenges that you found too difficult to overcome? If so, could you provide an example?

Participant 5: The second level was really hard for me. I just couldn't get away from the boss guy quick enough without messing up and landing in the spikes or just until he caught up with me.

Interviewer: Are there any changes or additions you think could be made to the game to make it more approachable for beginners like yourself?

Participant 5: I think the tutorial really helped and let me get used to how to shoot the bullets and kill the enemies. Maybe just getting more information on that would have helped me more, but I think most of it comes with practice.

Interviewer: What aspects of the game did you enjoy the most, and what do you think contributed to your overall enjoyment?

Participant 5: I enjoyed the sense of accomplishment that came from completing each level and overcoming the various challenges. The game's visual design and sound effects also contributed to creating an engaging and immersive experience for me, despite my lack of gaming experience.

Participant 6

Questionnaire Answers for Participant 6

Question 1: How experienced are you with playing video games?

- Intermediate

Question 2: How familiar are you with 2D platformer games?

- Very Familiar

Question 3: How would you rate the overall game design and visuals?

- Good

Question 4: How did you find the game's difficulty level?

- Just right

Question 5: On a scale of 1 to 4, how well do you agree with this statement? "The game controls were intuitive and easy to use".

- 4 (Strongly agree)

Question 6: On a scale of 1 to 4, do you feel that the invisibility feature was necessary to complete the game levels?

- 3

Question 7: On a scale of 1 to 4, did the uniqueness of the invisibility feature make the game more enjoyable?

- 4 (Strongly agree)

Question 8: Did you encounter any issues or problems while playing the game?

- Enemy AI could be more challenging and responsive to player actions.

Question 9: On a scale of 1 to 4, overall, did you enjoy playing the game?

- 4 (Yes, significantly)

Participant 6 Interview Transcript:

Interviewer: What are your overall impressions of the game? Did you find it enjoyable to play?

Participant 6: I thought the game was well-designed and fun to play. The mix of platforming, shooting, and the invisibility part made it enjoyable.

Interviewer: How would you rate the difficulty of the game? Did you find it too easy, too hard, or just right?

Participant 6: For me, the difficulty was hard at first, but the longer you play the easier it gets. The second level was much harder than the first I think, but it was also good once you get used to how the big enemy guy moves.

Interviewer: What are your thoughts on the invisibility feature? How did it impact your gameplay experience?

Participant 6: I found the invisibility feature to be a unique thing to have in the game. It meant that you could think differently when compared to a normal 2D platformer. I think maybe if the player could only be invisible for a certain amount of time, that would make you think more about when to use it or not.

Interviewer: How did you find the enemy AI behaviour? Did it adapt well to the invisibility feature?

Participant 6: The enemy AI seemed to change with the invisibility feature, but I think it could be made harder. It didn't take me long to figure out how to plan my routes through them.

Interviewer: You mentioned that the enemy AI could be more challenging. Can you suggest any specific improvements to the enemy AI that would make them more difficult to exploit?

Participant 6: I think the AI should be more responsive to player actions, especially when using invisibility. They could become more suspicious, like changing how they move or calling for more enemies to come in when they detect you or something like that.

Interviewer: Since you found the game a bit easy, would you prefer to see more diverse types of enemies, or should the focus be on making the existing enemies more challenging?

Participant 6: I think probably both. Having new enemy types with new abilities could add variety, and it would make the enemies already there harder to fight by increasing how unpredictable they are.

Interviewer: You mentioned that adding a cooldown or energy meter for the invisibility feature could make it more strategic. Can you elaborate on how this would impact the overall gameplay experience?

Participant 6: Well, I think players would need to think more about when to use the feature. They would need to weigh up the risks of using it at certain times so the game is more strategic. This would also mean people can't overuse the invisible power.

Interviewer: Considering your experience with other games already, are there any additional features or elements that you think could enhance the overall gameplay experience of this 2D platformer?

Participant 6: I think maybe like a progression system where you can unlock abilities or upgrades for the player could make it more. Also, having things like collectables could encourage the player to explore around the game levels more.

Appendix E: Requirements

Functional Requirements

- Develop a 2D platformer game with a unique set of innovative mechanics that distinguish it from traditional platformers.
- Implement a character controller that allows for smooth movement, including walking, running, jumping, and interacting with objects in the game world.
- Create a series of levels with varying degrees of difficulty, utilising innovative mechanics to provide a fresh and engaging gameplay experience.
- Integrate a pathfinding system, using Aron Granberg's A* Pathfinding Project, to enable enemies to navigate the game world intelligently.
- Design a user-friendly interface for the main menu, settings, and in-game HUD, allowing players to navigate and interact with the game easily.
- Create diverse obstacles, enemies, and collectables to challenge and engage the player throughout the game.

Non-functional Requirements

- Performance: The game should maintain a stable frame rate, minimising lag and providing a smooth gameplay experience.
- Usability: The game controls should be intuitive and responsive, allowing players of various skill levels to engage with the game effectively.
- Scalability: The game's design should allow for easy addition or modification of levels, mechanics, and other content in the future.
- Maintainability: The code should be well-organised, modular, and easy to understand, allowing for potential updates or fixes in the future.
- Quality: The game should be polished and free of major bugs or issues, ensuring a satisfying experience for the players.
- Aesthetics: The game should feature appealing visual and audio design, contributing to an immersive and enjoyable gameplay experience.

Appendix F: My Scripts

AudioManager.cs (1.1)

```csharp
/*
 * AudioManager.cs
 * Author: William Dibble
 * Date: 24-04-2023
 *
 * This script is responsible for managing the audio in the game.
 *
 * It contains audio source and audio clip variables for sound effects and background
music,
 * as well as methods for playing each sound effect and playing music for the current
scene.
 */

using UnityEngine;
using UnityEngine.SceneManagement;
using System.Collections;

public class AudioManager : MonoBehaviour
{
    // Declare Audio Source and Audio Clip variables for the sound effects
    public AudioSource audioSource;
    public AudioClip blowSound;
    public AudioClip cinematicSound;
    public AudioClip cinematicSound2;
    public AudioClip coin1Sound;
    public AudioClip coin2Sound;
    public AudioClip collectSound;
    public AudioClip curiousSound;
    public AudioClip dotSound;
    public AudioClip explosionSound;
    public AudioClip fallImpactSound;
    public AudioClip hit1Sound;
    public AudioClip hit2Sound;
    public AudioClip hit3Sound;
    public AudioClip jump1Sound;
    public AudioClip jump2Sound;
    public AudioClip bigJumpSound;
    public AudioClip levelUpSound;
    public AudioClip machineSound;
    public AudioClip questionSound;
    public AudioClip select1Sound;
    public AudioClip select2Sound;
    public AudioClip select3Sound;
    public AudioClip select4Sound;
    public AudioClip select5Sound;
    public AudioClip slideSound;
    public AudioClip textSound;
    public AudioClip invisible1Sound;
    public AudioClip invisible2Sound;
    public AudioClip fire1;

    // Declare Audio Source and Audio Clip variables for the background music
    public AudioSource musicSource;
    public AudioClip startSceneMusic;
    public AudioClip level1Music;
    public AudioClip level2Music;
    public AudioClip level3Music;
    public AudioClip level4Music;
    public AudioClip finalScoreScreenMusic;

    public float musicFadeTime = 1f;

    private Coroutine fadeCoroutine;

    private void Awake()
    {
        // Add audio sources if they are not assigned
        if (audioSource == null)
        {
            audioSource = gameObject.AddComponent<AudioSource>();
        }
        if (musicSource == null)
        {
            musicSource = gameObject.AddComponent<AudioSource>();
            musicSource.loop = true;
        }

        // Register the OnSceneLoaded event
        SceneManager.sceneLoaded += OnSceneLoaded;
    }

    private void OnDestroy()
    {
        // Unregister the OnSceneLoaded event
        SceneManager.sceneLoaded -= OnSceneLoaded;
    }

    // Method called when a new scene is loaded
    private void OnSceneLoaded(Scene scene, LoadSceneMode mode)
    {
        PlaySceneMusic();
    }

    private void Start()
    {
        // Play the music for the current scene
        PlaySceneMusic();
    }

    // Methods to play each sound effect
    public void PlayBlowSound() { PlaySound(blowSound); }
    public void PlayCinematicSound() { PlaySound(cinematicSound); }
    public void PlayCinematicSound2() { PlaySound(cinematicSound2); }
    public void PlayCoin1Sound() { PlaySound(coin1Sound); }
    public void PlayCoin2Sound() { PlaySound(coin2Sound); }
    public void PlayCollectSound() { PlaySound(collectSound); }
    public void PlayCuriousSound() { PlaySound(curiousSound); }
    public void PlayDotSound() { PlaySound(dotSound); }
    public void PlayExplosionSound() { PlaySound(explosionSound); }
    public void PlayFallImpactSound() { PlaySound(fallImpactSound); }
    public void PlayHit1Sound() { PlaySound(hit1Sound); }
    public void PlayHit2Sound() { PlaySound(hit2Sound); }
    public void PlayHit3Sound() { PlaySound(hit3Sound); }
    public void PlayJump1Sound() { PlaySound(jump1Sound); }
    public void PlayJump2Sound() { PlaySound(jump2Sound); }
    public void PlayBigJumpSound() { PlaySound(bigJumpSound); }
    public void PlayLevelUpSound() { PlaySound(levelUpSound); }
    public void PlayMachineSound() { PlaySound(machineSound); }
    public void PlayQuestionSound() { PlaySound(questionSound); }
    public void PlaySelect1Sound() { PlaySound(select1Sound); }
    public void PlaySelect2Sound() { PlaySound(select2Sound); }
    public void PlaySelect3Sound() { PlaySound(select3Sound); }
    public void PlaySelect4Sound() { PlaySound(select4Sound); }
    public void PlaySelect5Sound() { PlaySound(select5Sound); }
    public void PlaySlideSound() { PlaySound(slideSound); }
    public void PlayTextSound() { PlaySound(textSound); }
    public void PlayInvisible1() { PlaySound(invisible1Sound); }
    public void PlayInvisible2() { PlaySound(invisible2Sound); }
    public void PlayFire1() { PlaySound(fire1); }

    // Method to play a specific sound effect
    private void PlaySound(AudioClip clip)
    {
        audioSource.PlayOneShot(clip);
    }

    // Method to play music for the current scene
    public void PlaySceneMusic()
    {
        string sceneName = SceneManager.GetActiveScene().name;
        AudioClip targetClip = null;

        // Select appropriate music based on scene name
        if (sceneName == "Start Scene")
        {
            targetClip = startSceneMusic;
        }
        else if (sceneName == "Level 1")
        {
            targetClip = level1Music;
        }
        else if (sceneName == "Level 2")
        {
            targetClip = level2Music;
        }
        else if (sceneName == "Level 3")
        {
            targetClip = level3Music;
        }
        else if (sceneName == "Level 4")
        {
            targetClip = level4Music;
        }
        else if (sceneName == "Final Score Screen")
        {
            targetClip = finalScoreScreenMusic;
        }

        // Fade out old music and play new music if target clip is different
        if (targetClip != null && musicSource.clip != targetClip)
        {
            if (fadeCoroutine != null)
            {
                StopCoroutine(fadeCoroutine);
            }
            fadeCoroutine = StartCoroutine(FadeAndPlayMusic(targetClip));
        }
    }

    // Coroutine to fade out current music and fade in new music
    private IEnumerator FadeAndPlayMusic(AudioClip clip)
    {
        // Fade out current music
        float startTime = Time.time;
        float startVolume = musicSource.volume;
        while (Time.time < startTime + musicFadeTime)
        {
            musicSource.volume = Mathf.Lerp(startVolume, 0, (Time.time - startTime) / 
musicFadeTime);
            yield return null;
        }
        musicSource.volume = 0;

        // Set new clip and start playing
        musicSource.clip = clip;
        musicSource.Play();

        // Fade in new music
        startTime = Time.time;
        while (Time.time < startTime + musicFadeTime)
        {
            musicSource.volume = Mathf.Lerp(0, startVolume, (Time.time - startTime) / 
musicFadeTime);
            yield return null;
        }
        musicSource.volume = startVolume;
    }
}
```

BatEnemyAI.cs (1.2)

```
/*
 * BatEnemyAI.cs
 * Author: William Dibble
 * Date: 24-04-2023
 *
 * This is the script for controlling the AI of the final boss bat in Level 4.
 *
 * The bat uses pathfinding to follow the player and has a health bar.
 *
 * It can also become invisible, and it moves at a slower speed when the player is 
invisible.
 */

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Pathfinding;
using UnityEngine.UI;

public class BatEnemyAI : MonoBehaviour
{
    public Transform target; // Reference to the target (the player)

    public float speed = 200f; // Speed at which the bat moves towards the target
    public float nextWaypointDistance = 3f; // Distance before the bat moves to the 
next waypoint
    public float verticalMovementSpeed = 0.5f; // Speed for the bat's vertical 
movement
    private float originalHealthBarWidth;

    Path path; // The path for the bat to follow
    int currentWaypoint = 0; // Index of the current waypoint the bat is moving
towards
    bool reachedEndOfPath = false; // Check if the bat has reached the end of the path
    public int health = 100; // Bat's health
    private int startingHealth = 0; // The bat's starting health

    private LevelManager levelManager; // Reference to the level manager script

    Seeker seeker; // Reference to the Seeker component (for pathfinding)
    Rigidbody2D rb; // Reference to the Rigidbody2D component
    private SpriteRenderer sprite; // Reference to the SpriteRenderer component
    private BoxCollider2D collision; // Reference to the BoxCollider2D component
    private AudioManager audioManager; // Reference to the AudioManager script

    [SerializeField] private Image healthBarImage; // Reference to the health bar UI
image
    Material material; // Reference to the bat's material (for invisibility)

    private bool batInvisible = false; // Flag to track if the bat is invisible
    private float fade = 1f; // Opacity value for the bat's sprite (1 is fully
visible, 0 is fully invisible)
    private float invisibilityDelay = 1f; // Delay before toggling the bat's
invisibility
    private float timeToToggleInvisibility; // Time when the bat should toggle its
invisibility
    private bool isChangingInvisibility = false; // Flag to track if the bat is in the
process of changing its invisibility

    // Start is called before the first frame update
    void Start()
    {
        seeker = GetComponent<Seeker>();
        rb = GetComponent<Rigidbody2D>();
        sprite = GetComponent<SpriteRenderer>();
        collision = GetComponent<BoxCollider2D>();
        audioManager = FindObjectOfType<AudioManager>();

        levelManager = FindObjectOfType<LevelManager>();

        // Get the material component of the bat
        material = GetComponent<SpriteRenderer>().material;

        // Repeatedly update the path to the target
        InvokeRepeating("UpdatePath", 0f, .5f);

        originalHealthBarWidth =
healthBarImage.GetComponent<RectTransform>().sizeDelta.x;

        startingHealth = health;
    }

    // Update the path to the target
    void UpdatePath()
    {
        float distanceToTarget = Vector2.Distance(transform.position,
target.position);

        if (seeker.IsDone())
        {
            seeker.StartPath(rb.position, target.position, OnPathComplete);
        }
    }
    // Set the new path if it was found without errors
    void OnPathComplete(Path p)
    {
        if (!p.error)
        {
            path = p;
            currentWaypoint = 0;
        }
    }

    void Update()
    {
        // Handle invisibility
        HandleInvisibility();
    }

    // FixedUpdate is called at a fixed interval and used for physics calculations
    void FixedUpdate()
    {
        if (path == null)
        {
            return;
        }

        float distanceToTarget = Vector2.Distance(transform.position,
target.position);

        if (distanceToTarget > 13f)
        {
            if (PlayerVisibility.GetInvisible() && distanceToTarget > 8f)
            {
                // Player is out of range, do not move towards them
                return;
            }
        }

        if (currentWaypoint >= path.vectorPath.Count)
        {
            reachedEndOfPath = true;
            return;
        }
        else
        {
            reachedEndOfPath = false;
        }

        // Calculate the direction and force to move the bat towards the target
        Vector2 direction = ((Vector2)path.vectorPath[currentWaypoint] -
rb.position).normalized;
        Vector2 force = speed * Time.deltaTime * direction;

        if (PlayerVisibility.GetInvisible())
        {
            force = (speed / 2) * Time.deltaTime * direction;
        }

        rb.AddForce(force);

        // Update the current waypoint if the bat is close enough
        float distance = Vector2.Distance(rb.position,
path.vectorPath[currentWaypoint]);
```

```
        if (distance < nextWaypointDistance)
        {
            currentWaypoint++;
        }


        // Determine which way the enemy should face
        float targetXPos = target.position.x;

        if (transform.position.x > targetXPos)
        {
            // Face left
            sprite.flipX = false;
        }
        else
        {
            // Face right
            sprite.flipX = true;
        }
    }

    // Handle the bat's behavior when the player is invisible
    private void HandleInvisibility()
    {
        if (!isChangingInvisibility && PlayerVisibility.GetInvisible() !=
batInvisible)
        {
            if (timeToToggleInvisibility <= Time.time)
            {
                StartCoroutine(ToggleInvisibilitySmoothly());
            }
        }
        else
        {
            timeToToggleInvisibility = Time.time + invisibilityDelay;
        }
    }

    public void TakeDamage(int damage)
    {
        health -= damage;
        UpdateHealthBar();
        audioManager.PlayHit2Sound();

        if (health <= 0)
        {
            Die();
            levelManager.LoadLevel("Final Score Screen");
        }
    }

    private void UpdateHealthBar()
    {
        float healthPercentage = (float)health / startingHealth;
        RectTransform healthBarRectTransform =
healthBarImage.GetComponent<RectTransform>();
        Vector2 sizeDelta = healthBarRectTransform.sizeDelta;
        sizeDelta.x = healthPercentage * originalHealthBarWidth;
        healthBarRectTransform.sizeDelta = sizeDelta;
    }
    void Die()
    {
        audioManager.PlayHit3Sound();
        audioManager.PlayExplosionSound();
        Destroy(gameObject);
    }

    void OnTriggerEnter2D(Collider2D other)
    {
        if (other.gameObject.CompareTag("Trap"))
        {
            // Ignore collisions between the enemy and the trap object
            Physics2D.IgnoreCollision(collision, other, true);
        }
    }

    void OnTriggerExit2D(Collider2D other)
    {
        if (other.gameObject.CompareTag("Trap"))
        {
            // Restore collisions between the enemy and the trap object
            Physics2D.IgnoreCollision(collision, other, false);
        }
    }

    private IEnumerator ToggleInvisibilitySmoothly()
    {
        isChangingInvisibility = true;
        float targetFade = batInvisible ? 1f : 0f;

        while (!Mathf.Approximately(fade, targetFade))
        {
            fade = Mathf.MoveTowards(fade, targetFade, Time.deltaTime);
            material.SetFloat("_Fade", fade);
            yield return null;
        }

        batInvisible = !batInvisible;
        isChangingInvisibility = false;
    }
}
```

BreakablePlatform.cs (1.3)

```
/*
 * BreakablePlatform.cs
 * Author: William Dibble
 * Date: 24-04-2023
 *
 * This script is attached to a breakable platform game object, and it allows the
platform to break when the player collides with it.
 *
 */

using System.Collections;
using UnityEngine;

public class BreakablePlatform : MonoBehaviour
{
    [SerializeField] private GameObject breakingPlatformPrefab; // Prefab for the
breaking platform animation
```

```
    [SerializeField] private float delayBeforeBreaking = 0.5f; // Time delay before
the platform breaks
    private AudioManager audioManager; // Reference to the AudioManager

    // Start is called before the first frame update
    void Start()
    {
        // Get a reference to the AudioManager object
        audioManager = FindObjectOfType<AudioManager>();
    }

    // OnCollisionEnter2D is called when the platform collides with another object
    private void OnCollisionEnter2D(Collision2D other)
    {
        // Check if the colliding object has the "Player" tag
        if (other.gameObject.CompareTag("Player"))
        {
            // Start the BreakPlatform coroutine
            StartCoroutine(BreakPlatform());
        }
    }

    // Coroutine to break the platform after a specified delay
    private IEnumerator BreakPlatform()
    {
        // Wait for the specified delay before breaking
        yield return new WaitForSeconds(delayBeforeBreaking);

        // Instantiate the breaking platform animation object
        Instantiate(breakingPlatformPrefab, transform.position, Quaternion.identity);
        // Play the fall impact sound effect
        audioManager.PlayFallImpactSound();

        // Destroy the breakable platform object
        Destroy(gameObject);
    }
}
```

CinemachineController.cs (1.4)

```
/*
 * TransitionImageController.cs
 * Author: William Dibble
 * Date: 24-04-2023
 *
 * This script is responsible for controlling the Cinemachine Virtual Camera
component, specifically its orthographic size.
 *
 * The orthographic size determines how much of the game world is visible to the
player,
 * with a smaller value showing less of the world and a larger value showing more.
 */

using Cinemachine;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class CinemachineController : MonoBehaviour
```

```
{
    private CinemachineVirtualCamera vcam;
    private float originalOrthoSize;
    [SerializeField] private float invisibleOrthoSize;

    // Start is called before the first frame update
    void Start()
    {
        // Get the Cinemachine Virtual Camera component and store the original
orthographic size
        vcam = GetComponent<CinemachineVirtualCamera>();
        originalOrthoSize = vcam.m_Lens.OrthographicSize;
    }

    // Update is called once per frame
    void Update()
    {
        // Get the current scene name
        string currentSceneName = SceneManager.GetActiveScene().name;

        // Return early if the current scene is Level 3 or Level 4, as no changes are
needed
        if (currentSceneName == "Level 3" || currentSceneName == "Level 4")
        {
            return;
        }

        // Get the player's visibility state
        bool isInvisible = PlayerVisibility.IsInvisible;

        // If the player is invisible and the camera's orthographic size is not set to
invisibleOrthoSize, interpolate towards it
        if (isInvisible && vcam.m_Lens.OrthographicSize != invisibleOrthoSize)
        {
            vcam.m_Lens.OrthographicSize = Mathf.Lerp(vcam.m_Lens.OrthographicSize,
invisibleOrthoSize, Time.deltaTime * 2);
        }
        // If the player is not invisible and the camera's orthographic size is not
set to the originalOrthoSize, interpolate towards it
        else if (!isInvisible && vcam.m_Lens.OrthographicSize != originalOrthoSize)
        {
            vcam.m_Lens.OrthographicSize = Mathf.Lerp(vcam.m_Lens.OrthographicSize,
originalOrthoSize, Time.deltaTime * 2);
        }
    }
}
```

CreditsMenu.cs (1.5)

```
/*
 * CreditsMenu.cs
 * Author: William Dibble
 * Date: 24-04-2023
 *
 * This script is responsible for controlling the Credits menu in the game.
 * It has a reference to the UI game object of the Credits menu, which can be toggled
on and off by the player.
 */

using UnityEngine;
using UnityEngine.SceneManagement;
```

```
public class CreditsMenu : MonoBehaviour
{
    public GameObject creditsMenuUI;

    private bool isPaused = false;

    // Reference to the Timer script
    private Timer timer;

    private LevelManager levelManager;

    private AudioManager audioManager;

    private void Start()
    {
        // Find the Timer object and get its Timer script component
        timer = FindObjectOfType<Timer>();

        levelManager = FindObjectOfType<LevelManager>();

        audioManager = FindObjectOfType<AudioManager>();
    }

    // Update is called once per frame
    void Update()
    {

    }

    // Resume the game and deactivate the pause menu
    public void Resume()
    {
        creditsMenuUI.SetActive(false);
        Time.timeScale = 1f;
        isPaused = false;
    }

    public void Credits()
    {
            // Toggle the pause menu based on the current state
            if (isPaused)
            {
                Resume();
            }
            else
            {
                Pause();
            }

        audioManager.PlaySelect1Sound();
        audioManager.PlaySelect3Sound();
    }

    // Pause the game and activate the pause menu
    void Pause()
    {
        creditsMenuUI.SetActive(true);
        Time.timeScale = 0f;
        isPaused = true;
    }

    // Load the main menu scene
    public void MainMenu()
    {
        Time.timeScale = 1f; // Reset time scale to 1 before quitting
        Destroy(timer.gameObject); // Destroy Timer as a new one is created in the
Start Scene
        SceneManager.LoadScene("Start Scene"); // Load the main menu scene

        audioManager.PlaySelect2Sound();
        audioManager.PlaySelect4Sound();
    }
}
```

DynamicStartScreen.cs (1.6)

```
/*
 * DynamicStartScreen.cs
 * Author: William Dibble
 * Date: 24-04-2023
 *
 * This script creates a parallax effect on a background layer based on the mouse
position.
 *
 * The script calculates the distance of the layer from its starting position and
adjusts the
 * parallax effect multiplier accordingly to create a sense of depth.
 *
 * The script uses the mouse coordinates on the screen to calculate the horizontal and
vertical movement of the layer
 * and moves the layer towards the target position using a smoothing effect.
 *
 * The script also has variables to control the strength of the parallax effect, the
smoothness of the movement,
 * the distance at which the parallax effect reduces to zero, and the parallax effect
factor for each layer.
 */

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class DynamicStartScreen : MonoBehaviour
{
    // Multiplier for the parallax effect strength
    public float parallaxEffectMultiplier = 0.1f;

    // Controls the smoothness of the movement
    public float smoothness = 5f;

    // The distance at which the parallax effect reduces to zero
    public float distanceEffectRange = 50f;

    // Factor for controlling the parallax effect for each layer (set in the
Inspector)
    public float parallaxLayerFactor = 1f;

    // The starting position of the background layer
    private Vector3 startPosition;

    // Width and height of the screen
    private float screenWidth;
    private float screenHeight;
    void Start()
    {
        // Store the initial position of the background layer
        startPosition = transform.position;

        // Get the screen dimensions
        screenWidth = Screen.width;
        screenHeight = Screen.height;
    }

    void Update()
    {
        // Get the mouse coordinates on the screen
        float mouseX = Input.mousePosition.x;
        float mouseY = Input.mousePosition.y;

        // Calculate the distance from the initial position
        float distance = Vector3.Distance(transform.position, startPosition);

        // Calculate the adjusted parallax effect multiplier based on the distance and
layer factor
        float adjustedMultiplier = Mathf.Lerp(parallaxEffectMultiplier, 0, distance /
distanceEffectRange) * parallaxLayerFactor;

        // Calculate the horizontal and vertical movement based on the mouse position
and adjusted multiplier
        float deltaX = (mouseX - screenWidth * 0.5f) * adjustedMultiplier;
        float deltaY = (mouseY - screenHeight * 0.5f) * adjustedMultiplier;

        // Determine the target position for the layer based on the calculated
movement
        Vector3 targetPosition = new Vector3(startPosition.x + deltaX, startPosition.y
+ deltaY, startPosition.z);

        // Smoothly move the layer towards the target position
        transform.position = Vector3.Lerp(transform.position, targetPosition,
smoothness * Time.unscaledDeltaTime);
    }
}
```

EnemyBossMovement.cs (1.7)

```
/*
 * EnemyBossMovement.cs
 * Author: William Dibble
 * Date: 24-04-2023
 *
 * This script controls the movement and behavior of the enemy boss in Level 2.
 * The boss will jump towards the player character and attempt to catch up with them.
 * If the boss catches up to the player, the player will be killed.
 */

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class EnemyBossMovement : MonoBehaviour
{
    public GameObject player;
    public GameObject bossHead; // Reference to the Boss Head object
    public float jumpForce = 15f;
    public float jumpFrequency = 2f;
    public float horizontalSpeed = 1f;
    public float catchUpDistance = 0.5f;
    public float rotationSpeed = 2f; // Added rotation speed variable
    private Rigidbody2D rb;
    private readonly float minYPosition = -1.26f;
    private bool playerDead = false;
    private AudioManager audioManager;

    // Initialize components and start jump routine
    private void Start()
    {
        rb = GetComponent<Rigidbody2D>();
        StartCoroutine(JumpRoutine());
        audioManager = FindObjectOfType<AudioManager>();
    }

    // FixedUpdate is used for physics-related updates
    private void FixedUpdate()
    {
        // Check if the boss has caught up with the player
        PlayerLife playerLife = player.GetComponent<PlayerLife>();
        if (Mathf.Abs(transform.position.x - player.transform.position.x) <=
catchUpDistance)
        {
            // If the player is not dead, kill the player
            if (!playerDead)
            {
                playerDead = true;
                playerLife.Die();
            }
        }

        // Limit the boss's vertical position
        if (transform.position.y < minYPosition)
        {
            rb.velocity = new Vector2(rb.velocity.x, 0);
            transform.position = new Vector2(transform.position.x, minYPosition);
        }
    }

    // Update the boss's head rotation to face the player
    private void Update()
    {
        // Make the Boss Head look at the player
        Vector3 direction = player.transform.position - bossHead.transform.position;
        float angle = Mathf.Atan2(direction.y, direction.x) * Mathf.Rad2Deg;
        Quaternion targetRotation = Quaternion.Euler(new Vector3(0, 0, angle));
        bossHead.transform.rotation = Quaternion.Lerp(bossHead.transform.rotation,
targetRotation, rotationSpeed * Time.deltaTime);
    }

    // Coroutine to handle the boss's jumping behavior
    private IEnumerator JumpRoutine()
    {
        while (true)
        {
            // Wait for a duration depending on the player's visibility state
            if (PlayerVisibility.IsInvisible)
            {
                yield return new WaitForSeconds(jumpFrequency * 0.75f);
            }
            else
            {
                yield return new WaitForSeconds(jumpFrequency);
            }
            // If the boss is on the ground, apply a force to make it jump
            if (transform.position.y <= minYPosition)
            {
                rb.velocity = new Vector2(0, 0); // Reset velocity before applying the
force
                transform.position = new Vector2(transform.position.x, minYPosition);
                rb.AddForce(new Vector2(horizontalSpeed, jumpForce),
ForceMode2D.Impulse);
                audioManager.PlayBigJumpSound();
            }
        }
    }
}
```

FallingCavePlatformDestroy.cs (1.8)

```
/*
 * FallingCavePlatformDestroy.cs
 * Author: William Dibble
 * Date: 24-04-2023
 *
 * This script is used to destroy a falling cave platform after its animation has
played out.
 *
 */

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class FallingCavePlatformDestroy : MonoBehaviour
{
    [SerializeField] private float delay = 0.3f;

    // Start is called before the first frame update
    void Start()
    {
        Destroy(gameObject, delay);
    }
}
```

FlyingEnemyAI.cs (1.9)

```
/*
 * FlyingEnemyAI.cs
 * Author: William Dibble
 * Date: 24-04-2023
 *
 * This script handles the behavior of a flying enemy AI in the game.
 * The enemy follows a target (the player) and drops TNT on them after a delay.
 * The script includes methods for updating the enemy's path, dropping TNT, handling
collisions with traps, and flipping the sprite based on the direction of movement.
 *
 */
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Pathfinding;

public class FlyingEnemyAI : MonoBehaviour
{
    public Transform target; // Target (the player) that the enemy will follow

    public float speed = 200f; // Enemy movement speed
    public float nextWaypointDistance = 3f; // Distance to the next waypoint before
switching to the next
    public float height = 3f; // Height above the target that the enemy will fly

    private float tntDropDelay = 3f; // Delay between dropping TNT
    private float lastTntDropTime; // Time the last TNT was dropped

    Path path; // Path the enemy will follow
    int currentWaypoint = 0; // Current waypoint along the path
    bool reachedEndOfPath = false; // Whether the enemy has reached the end of the
path

    private Seeker seeker; // Seeker component for pathfinding
    private Rigidbody2D rb; // Rigidbody2D component for physics
    private SpriteRenderer sprite; // SpriteRenderer component for rendering and
flipping sprites
    private BoxCollider2D collision; // BoxCollider2D component for collision
detection
    private AudioManager audioManager; // AudioManager to play sound effects

    [SerializeField] private Transform dropPoint; // Transform where the TNT will be
dropped
    [SerializeField] private GameObject TNTPrefab; // TNT prefab to be instantiated

    void Start()
    {
        seeker = GetComponent<Seeker>();
        rb = GetComponent<Rigidbody2D>();
        sprite = GetComponent<SpriteRenderer>();
        collision = GetComponent<BoxCollider2D>();
        audioManager = FindObjectOfType<AudioManager>();

        // Repeatedly update the path every 0.5 seconds
        InvokeRepeating("UpdatePath", 0f, .5f);
    }

    // Function to instantiate and drop the TNT
    void DropTNT()
    {
        Instantiate(TNTPrefab, dropPoint.position, dropPoint.rotation);

        // If the player is invisible, spawn two additional TNTs
        if (PlayerVisibility.GetInvisible())
        {
            Instantiate(TNTPrefab, dropPoint.position + Vector3.left * 1.25f +
Vector3.up * 0.33f, dropPoint.rotation);
            Instantiate(TNTPrefab, dropPoint.position + Vector3.right * 1.25f +
Vector3.up * 0.33f, dropPoint.rotation);
        }
        audioManager.PlayMachineSound(); // Play sound effect when dropping TNT
    }
    // Function to update the enemy's path
    void UpdatePath()
    {
        float distanceToTarget = Vector2.Distance(transform.position,
target.position);
        float maxDistance = PlayerVisibility.GetInvisible() ? 8f : 13f;

        // If the player is too far away, don't update the path
        if (distanceToTarget > maxDistance)
        {
            return;
        }

        // If the pathfinding calculation is complete, start a new path to the
modified target position
        if (seeker.IsDone())
        {
            Vector3 modifiedTargetPosition = target.position + Vector3.up * height +
Vector3.right * 1f;
            seeker.StartPath(rb.position, modifiedTargetPosition, OnPathComplete);
        }
    }

    // Function to handle path completion
    void OnPathComplete(Path p)
    {
        if (!p.error)
        {
            path = p;
            currentWaypoint = 0;
        }
    }

    void FixedUpdate()
    {
        if (path == null)
        {
            return;
        }

        float distanceToTarget = Vector2.Distance(transform.position,
target.position);

        // If the player is out of range, do not move towards them
        if (distanceToTarget > 13f)
        {
            if (PlayerVisibility.GetInvisible() && distanceToTarget > 8f)
            {
                // Player is out of range, do not move towards them
                return;
            }
        }

        // If the enemy has reached the end of the path, stop moving
        if (currentWaypoint >= path.vectorPath.Count)
        {
            reachedEndOfPath = true;
            return;
        }
        else
        {
            reachedEndOfPath = false;
        }

        Vector2 direction = ((Vector2)path.vectorPath[currentWaypoint] - 
rb.position).normalized;
        Vector2 force = direction * speed * Time.deltaTime;

        rb.AddForce(force);

        float distance = Vector2.Distance(rb.position, 
path.vectorPath[currentWaypoint]);

        if (distance < nextWaypointDistance)
        {
            currentWaypoint++;
        }

        // Check if enemy X position matches player X position
        if (Mathf.Abs(transform.position.x - target.position.x) < 0.5f)
        {
            // Check if the delay time for dropping TNT has elapsed
            if (Time.time - lastTntDropTime > tntDropDelay)
            {
                // Drop TNT
                DropTNT();

                // Update last TNT drop time
                lastTntDropTime = Time.time;
            }
        }

        // Store the x position of the target
        float targetXPos = target.position.x;

        // Determine which way the enemy should face
        if (transform.position.x > targetXPos)
        {
            // Face left
            sprite.flipX = true;
        }
        else
        {
            // Face right
            sprite.flipX = false;
        }
    }

    void OnTriggerEnter2D(Collider2D other)
    {
        if (other.gameObject.CompareTag("Trap"))
        {
            // Ignore collisions between the enemy and the trap object
            Physics2D.IgnoreCollision(collision, other, true);
        }
    }

    void OnTriggerExit2D(Collider2D other)
    {
        if (other.gameObject.CompareTag("Trap"))
        {
            // Restore collisions between the enemy and the trap object
            Physics2D.IgnoreCollision(collision, other, false);
        }
    }
}
```

InvisiblePlatform.cs (1.10)

```
/*
 * InvisiblePlatform.cs
 * Author: William Dibble
 * Date: 24-04-2023
 *
 * This script handles the logic for making an invisible platform visible or invisible
depending on the player's visibility state.
 * It includes methods for checking the player's visibility state and updating the
platform's visibility status.
 * It also includes logic for enabling and disabling the platform's collider and
updating its tag.
 */

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class InvisiblePlatform : MonoBehaviour
{
    Material material;

    public bool alreadyInvisible;
    private bool isDissolving = false;
    private float fade = 1f;

    private Collider2D platformCollider;
    private bool collisionEnabled = true;
    private bool collisionDisabled = false;

    // Start is called before the first frame update
    void Start()
    {
        // Get the material and collider components of the platform
        material = GetComponent<SpriteRenderer>().material;
        platformCollider = GetComponent<Collider2D>();

        // Set the platform to be invisible and disable its collider if
alreadyInvisible is true
        if (alreadyInvisible)
        {
            fade = 0f;
            material.SetFloat("_Fade", fade);
            platformCollider.enabled = false;
            collisionEnabled = false;
            collisionDisabled = true;

            if (CompareTag("Trap"))
            {
                tag = "Untagged";
            }
        }
    }

    // Update is called once per frame
    void Update()
    {
        // Check if the player's invisibility state is different from the platform's 
initial state
        if (PlayerVisibility.GetInvisible() != alreadyInvisible)
        {
            if (!isDissolving)
            {
                isDissolving = true;
            }
        }
        else
        {
            if (isDissolving)
            {
                isDissolving = false;
            }
        }

        // Handle platform turning invisible
        if (isDissolving)
        {
            fade -= Time.deltaTime;

            if (fade <= 0f)
            {
                fade = 0f;
            }

            // Disable the platform's collider and remove the "Trap" tag as soon as it 
starts turning invisible
            if (!collisionDisabled)
            {
                platformCollider.enabled = false;
                collisionEnabled = false;
                collisionDisabled = true;

                if (CompareTag("Trap"))
                {
                    tag = "Untagged";
                }
            }

            material.SetFloat("_Fade", fade);
        }
        // Handle platform turning visible
        else if (!isDissolving)
        {
            fade += Time.deltaTime;

            if (fade >= 1f)
            {
                fade = 1f;
            }

            // Enable the platform's collider and add the "Trap" tag as soon as it 
starts turning visible
            if (!collisionEnabled)
            {
                platformCollider.enabled = true;
                collisionEnabled = true;
                collisionDisabled = false;
                if (gameObject.name.Contains("Spikes"))
                {
                    tag = "Trap";
                }

                if (gameObject.name.Contains("Blade"))
                {
                    tag = "Trap";
                }
            }

            material.SetFloat("_Fade", fade);
        }
    }
}
```

JumpingEnemyAI.cs (1.11)

```
/*
 * JumpingEnemyAI.cs
 * Author: William Dibble
 * Date: 24-04-2023
 *
 * This script is for controlling the behavior of a jumping enemy in the game.
 * The enemy is designed to jump towards a target and damage the player on contact.
 * The script uses pathfinding to calculate and update the enemy's path towards the
target,
 * but also uses a jump trajectory instead of a path if the enemy is on the ground and
within a certain distance from the target.
 */

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Pathfinding;

// Define JumpingEnemyAI class
public class JumpingEnemyAI : MonoBehaviour
{
    // Declare public variables
    public Transform target; // Target for the enemy to follow
    public Transform enemyGraphics; // Transform for the enemy's sprite
    public float jumpDelay = 2f; // Time delay between jumps
    public float jumpPower; // Jump power for the enemy
    public int health = 100; // Health value of the enemy

    // Declare private variables
    private Vector2 jumpStartPosition; // Starting position of the jump
    private Vector2 jumpTargetPosition; // Target position of the jump
    private Rigidbody2D rb; // Rigidbody2D component of the enemy
    private Seeker seeker; // Seeker component for pathfinding
    private BoxCollider2D collision; // BoxCollider2D component of the enemy
    private SpriteRenderer sprite; // SpriteRenderer component of the enemy
    private float distance; // Distance between the enemy and the target
    private AudioManager audioManager; // AudioManager component for playing sounds

    [SerializeField] private LayerMask onGroundMask; // LayerMask to check if the
enemy is grounded

    // Declare public float variable
    public float nextWaypointDistance = 3f; // Distance to the next waypoint
    // Start is called before the first frame update
    void Start()
    {
        // Assign component values
        rb = GetComponent<Rigidbody2D>();
        seeker = GetComponent<Seeker>();
        collision = GetComponent<BoxCollider2D>();
        sprite = GetComponent<SpriteRenderer>();
        audioManager = FindObjectOfType<AudioManager>();

        // Repeatedly call UpdatePath function
        InvokeRepeating("UpdatePath", 0f, 0.5f);
    }

    // UpdatePath function calculates and updates enemy's path towards target
    void UpdatePath()
    {
        distance = Vector2.Distance(target.position, rb.position);
        // Check if seeker is done and enemy is on ground
        if (seeker.IsDone() && onGround() && !PlayerVisibility.GetInvisible() &&
distance < 10f)
        {
            // Calculate jump trajectory instead of path
            jumpStartPosition = rb.position;
            jumpTargetPosition = target.position;
            jumpTargetPosition.y += 10.5f; // jump above target position
            float gravity = Physics2D.gravity.y;
            float timeToReachTarget = Mathf.Sqrt(-2f * jumpTargetPosition.y / gravity)
+ Mathf.Sqrt(2f * jumpStartPosition.y / gravity);

            // Check if timeToReachTarget is valid
            if (timeToReachTarget > 0f)
            {
                // Calculate velocity and make enemy jump
                Vector2 velocity = (jumpTargetPosition - jumpStartPosition) *
(timeToReachTarget);
                velocity.y = Mathf.Abs(velocity.y);
                if (velocity.x > 19)
                {
                    velocity.x = 19f;
                }
                else if (velocity.x < -19)
                {
                    velocity.x = -19f;
                }
                rb.velocity = velocity;
                audioManager.PlayBigJumpSound();
            }
            else // Handle invalid timeToReachTarget
            {
                Debug.LogWarning("Invalid timeToReachTarget: " + timeToReachTarget);
            }

            // Store the x position of the target
            float targetXPos = target.position.x;

            // Determine which way the enemy should face
            if (transform.position.x > targetXPos)
            {
                // Face left
                sprite.flipX = true;
            }
            else
            {
                // Face right
                sprite.flipX = false;
            }
        }
    }

    // Check if the enemy is on the ground
    private bool onGround()
    {
        return Physics2D.BoxCast(collision.bounds.center, collision.bounds.size, 0f, 
Vector2.down, .1f, onGroundMask);
    }

    public void TakeDamage(int damage)
    {
        health -= damage;

        audioManager.PlayHit2Sound();

        if (health <= 0)
        {
            Die();
        }
    }

    void Die()
    {
        audioManager.PlayHit1Sound();
        Destroy(gameObject);
    }

    void OnTriggerEnter2D(Collider2D other)
    {
        if (other.gameObject.CompareTag("Trap"))
        {
            // Ignore collisions between the enemy and the trap object
            Physics2D.IgnoreCollision(collision, other, true);
        }
    }

    void OnTriggerExit2D(Collider2D other)
    {
        if (other.gameObject.CompareTag("Trap"))
        {
            // Restore collisions between the enemy and the trap object
            Physics2D.IgnoreCollision(collision, other, false);
        }
    }
}
```

LevelManager.cs (1.12)

```
/*
 * LevelManager.cs
 * Author: William Dibble
 * Date: 24-04-2023
 *
 * This script manages the loading of levels and transition effects between them.
 *
 */

using System.Collections;
using UnityEngine;
using UnityEngine.SceneManagement;

public class LevelManager : MonoBehaviour
{
    public Timer timer; // Reference to the Timer script
    public TransitionImageController transitionImageController; // Reference to the
TransitionImageController script

    private void Start()
    {
        // Add a delegate method to be called when a new scene is loaded
        SceneManager.sceneLoaded += OnSceneLoaded;
    }

    private void OnDestroy()
    {
        // Remove the delegate method when the object is destroyed
        SceneManager.sceneLoaded -= OnSceneLoaded;
    }

    // This method is called when a new scene is loaded
    private void OnSceneLoaded(Scene scene, LoadSceneMode mode)
    {
        // Start the timer when Level 1 is loaded
        if (scene.name == "Level 1")
        {
            timer.StartTimer();
        }

        // Trigger the fade-out effect when the scene loads
        transitionImageController.FadeOut();
    }

    // Load the specified level with a transition effect
    public void LoadLevel(string levelName)
    {
        // Trigger the fade-in effect and wait for it to complete before loading a new
scene
        StartCoroutine(LoadLevelWithTransition(levelName));
    }

    // Coroutine to load the level with a fade-in transition
    private IEnumerator LoadLevelWithTransition(string levelName)
    {
        // Start the fade-in transition
        transitionImageController.FadeIn();

        // Wait for the transition to complete
        yield return new WaitForSeconds(transitionImageController.transitionTime);

        // Load the specified level
        SceneManager.LoadScene(levelName);
    }

    // Stop the timer
    public void StopTimer()
    {
        timer.StopTimer();
    }

}
```

NextLevelTrigger.cs (1.13)

```
/*
 * NextLevelTrigger.cs
 * Author: William Dibble
 * Date: 24-04-2023
 *
 * This script is attached to a trigger collider that loads the next level when the
player enters it.
 *
 */

using UnityEngine;
using UnityEngine.SceneManagement;

public class NextLevelTrigger : MonoBehaviour
{
    [SerializeField] private string nextLevelName;
    private LevelManager levelManager;

    // Reference to the Timer script
    private Timer timer;

    private void Start()
    {
        // Find the LevelManager object and get its LevelManager script component
        levelManager = FindObjectOfType<LevelManager>();

        // Find the Timer object and get its Timer script component
        timer = FindObjectOfType<Timer>();
    }

    // OnTriggerEnter2D is called when the player enters the trigger collider
    private void OnTriggerEnter2D(Collider2D other)
    {
        // Check if the object entering the trigger is tagged as "Player"
        if (other.CompareTag("Player"))
        {
            // Save the current bullet counts to the appropriate level's bullet count
            if (SceneManager.GetActiveScene().name == "Level 1")
            {
                int[] bulletCounts = { timer.GetBullet1Count(),
timer.GetBullet2Count(), timer.GetBullet3Count() };
                timer.SetBulletLevel2Count(bulletCounts);
            }

            else if (SceneManager.GetActiveScene().name == "Level 2")
            {
                int[] bulletCounts = { timer.GetBullet1Count(),
timer.GetBullet2Count(), timer.GetBullet3Count() };
                timer.SetBulletLevel3Count(bulletCounts);
            }

            else if (SceneManager.GetActiveScene().name == "Level 3")
            {
                int[] bulletCounts = { timer.GetBullet1Count(),
timer.GetBullet2Count(), timer.GetBullet3Count() };
```

```
                timer.SetBulletLevel4Count(bulletCounts);
            }

            levelManager.LoadLevel(nextLevelName);
        }
    }
}
```

ParallaxBackground.cs (1.14)

```
/*
 * ParallaxBackground.cs
 * Author: William Dibble
 * Date: 24-04-2023
 *
 * This script is attached to a GameObject representing the parallax background.
 * It initializes and manages the parallax layers, which are its child GameObjects
with ParallaxLayer components.
 * The script listens to the onCameraTranslate event from the ParallaxCamera script,
 * updating the positions of the parallax layers according to the camera's movement.
 * This script is responsible for creating the overall parallax effect by coordinating
the movements of all the individual layers.
 *
 */

using System.Collections.Generic;
using UnityEngine;

[ExecuteInEditMode]
public class ParallaxBackground : MonoBehaviour
{
    public ParallaxCamera parallaxCamera;
    List<ParallaxLayer> parallaxLayers = new List<ParallaxLayer>();

    void Start()
    {
        // Set parallaxCamera to the main camera's ParallaxCamera component if not
assigned
        if (parallaxCamera == null)
            parallaxCamera = Camera.main.GetComponent<ParallaxCamera>();

        // Subscribe to onCameraTranslate event
        if (parallaxCamera != null)
            parallaxCamera.onCameraTranslate += UpdateLayerPositions;

        // Initialize parallax layers
        InitializeLayers();
    }

    // Initialize parallaxLayers list with child objects containing ParallaxLayer
components
    void InitializeLayers()
    {
        parallaxLayers.Clear();
        for (int i = 0; i < transform.childCount; i++)
        {
            ParallaxLayer layer = transform.GetChild(i).GetComponent<ParallaxLayer>();

            if (layer != null)
            {
                layer.name = "Layer_" + i;
                parallaxLayers.Add(layer);
            }
        }
    }

    // Update the position of each parallax layer based on the camera's movement
    void UpdateLayerPositions(Vector2 delta)
    {
        foreach (ParallaxLayer layer in parallaxLayers)
        {
            layer.Move(delta);
        }
    }
}
```

ParallaxCamera.cs (1.15)

```
/*
 * ParallaxCamera.cs
 * Author: William Dibble
 * Date: 24-04-2023
 *
 * This script is attached to the main camera of the game.
 * It detects the camera's movement and triggers the onCameraTranslate event,
 * passing the change in position (delta) as a parameter. The event is used to inform
other
 * scripts (like ParallaxLayer) about the camera's movement, so they can adjust their
behavior accordingly.
 *
 */

using UnityEngine;

[ExecuteInEditMode]
public class ParallaxCamera : MonoBehaviour
{
    // Declare a delegate to handle camera movement events
    public delegate void ParallaxCameraDelegate(Vector2 deltaMovement);

    // Create an event to handle camera movement
    public ParallaxCameraDelegate onCameraTranslate;

    // Store the old position of the camera
    private Vector2 oldPosition;

    // Initialize the old position with the camera's position
    void Start()
    {
        oldPosition = transform.position;
    }

    // Check if the camera's position has changed and trigger the onCameraTranslate
event
    void FixedUpdate()
    {
        if ((Vector2)transform.position != oldPosition)
        {
            if (onCameraTranslate != null)
            {
                Vector2 delta = oldPosition - (Vector2)transform.position;
                onCameraTranslate(delta);
            }
            oldPosition = transform.position;
        }
    }
}
```

ParallaxLayer.cs (1.16)

```
/*
 * ParallaxLayer.cs
 * Author: William Dibble
 * Date: 24-04-2023
 *
 * This script is attached to individual layers of the parallax background.
 * It listens to the onCameraTranslate event from the ParallaxCamera script and moves
the layer
 * accordingly based on the camera's movement and the parallax factor. The parallax
factor is used
 * to create a depth effect, making some layers move slower or faster, giving the
illusion of a 3D environment.
 *
 */

using UnityEngine;

[ExecuteInEditMode]
public class ParallaxLayer : MonoBehaviour
{
    // Parallax factor to control the speed of the layer movement
    public float parallaxFactor;

    // Move the layer based on the camera's movement and parallax factor
    public void Move(Vector2 delta)
    {
        // Calculate the new position of the layer based on the camera movement and
parallax factor
        Vector3 newPos = transform.localPosition;
        newPos.x -= delta.x * parallaxFactor;
        newPos.y -= delta.y * parallaxFactor;

        // Update the layer's position
        transform.localPosition = newPos;
    }
}
```

PlatformStick.cs (1.17)

```
/*
 * PlatformStick.cs
 * Author: William Dibble
 * Date: 24-04-2023
 *
 * This script makes the player stick to the platform they are touching by making the
player a child of the platform,
 * and de-parents the player when they leave the platform.
 */

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PlatformStick : MonoBehaviour
{
    // If the player is touching the platform, make the player a child of the platform
so that they stick to it
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.gameObject.CompareTag("Player"))
        {
            collision.gameObject.transform.SetParent(transform);
        }
    }

    // When the player leaves the platform, de-parent it from the platform
    private void OnTriggerExit2D(Collider2D collision)
    {
        collision.gameObject.transform.SetParent(null);
    }
}
```

PlayerLife.cs (1.18)

```
/*
 * PlayerLife.cs
 * Author: William Dibble
 * Date: 24-04-2023
 *
 * This script handles the player's life, including detecting collisions with traps,
handling the player's death, and restarting the level.
 * It also sets the player's bullet counts based on the current level.
 */

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class PlayerLife : MonoBehaviour
{
    private Rigidbody2D body;
    private Animator anim;
    private BoxCollider2D collide;

    // Reference to the Timer script
    private Timer timer;

    // Reference to the Audio Manager script
    private AudioManager audioManager;

    private LevelManager levelManager;

    // Invincibility toggle for debugging purposes
    [SerializeField] private bool invincible = false;

    private void Start()
    {
        // Get the necessary components from the player object
        body = GetComponent<Rigidbody2D>();
        anim = GetComponent<Animator>();
        collide = GetComponent<BoxCollider2D>();
        // Find the Timer object and get its Timer script component
        timer = FindObjectOfType<Timer>();

        // Find the Audio object and get its Audio Manager script component
        audioManager = FindObjectOfType<AudioManager>();

        levelManager = FindObjectOfType<LevelManager>();

        // Set bullet counts based on the current level
        if (SceneManager.GetActiveScene().name == "Level 0")
        {
            timer.SetBullet1Count(0);
            timer.SetBullet2Count(0);
            timer.SetBullet3Count(0);
        }
        else if (SceneManager.GetActiveScene().name == "Level 1")
        {
            timer.SetBullet1Count(timer.GetBulletLevel1Count()[0]);
            timer.SetBullet2Count(timer.GetBulletLevel1Count()[1]);
            timer.SetBullet3Count(timer.GetBulletLevel1Count()[2]);
        }
        else if (SceneManager.GetActiveScene().name == "Level 2")
        {
            timer.SetBullet1Count(timer.GetBulletLevel2Count()[0]);
            timer.SetBullet2Count(timer.GetBulletLevel2Count()[1]);
            timer.SetBullet3Count(timer.GetBulletLevel2Count()[2]);
        }
        else if (SceneManager.GetActiveScene().name == "Level 3")
        {
            timer.SetBullet1Count(timer.GetBulletLevel3Count()[0]);
            timer.SetBullet2Count(timer.GetBulletLevel3Count()[1]);
            timer.SetBullet3Count(timer.GetBulletLevel3Count()[2]);
        }
        else if (SceneManager.GetActiveScene().name == "Level 4")
        {
            timer.SetBullet1Count(timer.GetBulletLevel4Count()[0]);
            timer.SetBullet2Count(timer.GetBulletLevel4Count()[1]);
            timer.SetBullet3Count(timer.GetBulletLevel4Count()[2]);
        }
    }

    // Detect collision with traps
    private void OnCollisionEnter2D(Collision2D collision)
    {
        if (!invincible && collision.gameObject.CompareTag("Trap"))
        {
            Die();
        }
    }

    // Handle player death
    public void Die()
    {
        // Play the death sound
        audioManager.PlayHit3Sound();

        // Make the player's Rigidbody static so they don't move
        body.bodyType = RigidbodyType2D.Static;

        // Trigger the death animation
        anim.SetTrigger("death");
        // Disable the player's BoxCollider
        collide.enabled = false;

        // Make the player visible if they were invisible
        if (PlayerVisibility.IsInvisible)
        {
            PlayerVisibility.SetInvisible(false);
        }
    }

    // Restart the level after death
    private void LevelRestart()
    {
        // Reload the current scene
        SceneManager.LoadScene(SceneManager.GetActiveScene().name);

        // Re-enable the player's BoxCollider
        collide.enabled = true;
    }
}
```

PlayerMovement.cs (1.19)

```
/*
 * PlayerMovement.cs
 * Author: William Dibble
 * Date: 24-04-2023
 *
 * This script is responsible for controlling the movement of the player character,
including horizontal movement, jumping, and double jumping.
 * The script also updates the player's animations based on their movement and state.
 */

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class PlayerMovement : MonoBehaviour
{
    // Components and variables needed for player movement and attributes
    private Rigidbody2D rb;
    private BoxCollider2D collision;
    private Animator anim;

    // Reference to the Timer script
    private Timer timer;

    private LevelManager levelManager;

    // Reference to the Audio Manager script
    private AudioManager audioManager;

    // Variables for double jump
    private bool canDoubleJump = false;
    private bool hasDoubleJumped = false;

    // Variables for ground detection
    [SerializeField] private LayerMask onGroundMask;
    [SerializeField] private GameObject teleportPoint;
```

```csharp
private float dirX = 0f;
[SerializeField] private float moveVelocity = 7f;
[SerializeField] private float jumpVelocity = 14f;

// Animation states for the player
private enum AnimationState { idle, running, jumping, falling }

// Start is called before the first frame update
void Start()
{
    // Get the necessary components
    rb = GetComponent<Rigidbody2D>();
    anim = GetComponent<Animator>();
    collision = GetComponent<BoxCollider2D>();

    // Find the Timer object and get its Timer script component
    timer = FindObjectOfType<Timer>();

    // Get the LevelManager script
    levelManager = FindObjectOfType<LevelManager>();

    // Find the Audio object and get its Audio Manager script component
    audioManager = FindObjectOfType<AudioManager>();

    // Enable double jump in certain levels
    if (SceneManager.GetActiveScene().name == "Level 3")
    {
        canDoubleJump = true;
    }

    else if (SceneManager.GetActiveScene().name == "Level 4")
    {
        canDoubleJump = true;
    }
}

// Update is called once per frame
void Update()
{
    // Get the input for horizontal movement
    dirX = Input.GetAxisRaw("Horizontal");
    rb.velocity = new Vector2(dirX * moveVelocity, rb.velocity.y);

    // Handle jumping and double jumping
    if (Input.GetButtonDown("Jump"))
    {
        if (onGround())
        {
            rb.velocity = new Vector2(rb.velocity.x, jumpVelocity);
            audioManager.PlayJump2Sound();
            hasDoubleJumped = false;
        }
        else if (canDoubleJump && !hasDoubleJumped)
        {
            rb.velocity = new Vector2(rb.velocity.x, jumpVelocity);
            audioManager.PlayJump2Sound();
            hasDoubleJumped = true;
        }
    }
```

```
        /*// Teleport the player to the teleport point when 'R' is pressed (DEV DEBUG
USE ONLY)
        if (Input.GetKeyDown(KeyCode.R))
        {
            transform.position = teleportPoint.transform.position;
        }*/


        // FOR DEMO PURPOSES, PRESSING 1, 2, 3, OR 4, WILL QUICK JUMP YOU TO WHICHEVER
LEVEL IS CHOSEN.
        // THIS FEATURE WOULD NOT BE INCLUDED IN THE PUBLISHED BUILD AND IS INTENDED
TO BE USED BY THE DEVELOPER OR WHEN BEING ASSESSED.
        // Skip to Level 1 and set bullet counts when '1' is pressed
        if (Input.GetKeyDown(KeyCode.Alpha1))
        {
            int[] bulletCounts = { 8, 8, 8 };
            timer.SetBulletLevel1Count(bulletCounts);
            levelManager.LoadLevel("Level 1");
        }

        // Skip to Level 2 and set bullet counts when '2' is pressed
        if (Input.GetKeyDown(KeyCode.Alpha2))
        {
            int[] bulletCounts = { 8, 8, 8 };
            timer.SetBulletLevel2Count(bulletCounts);
            levelManager.LoadLevel("Level 2");
        }

        // Skip to Level 3 and set bullet counts when '3' is pressed
        if (Input.GetKeyDown(KeyCode.Alpha3))
        {
            int[] bulletCounts = { 8, 8, 8 };
            timer.SetBulletLevel3Count(bulletCounts);
            levelManager.LoadLevel("Level 3");
        }

        // Skip to Level 4 and set bullet counts when '4' is pressed
        if (Input.GetKeyDown(KeyCode.Alpha4))
        {
            int[] bulletCounts = { 8, 8, 8 };
            timer.SetBulletLevel4Count(bulletCounts);
            levelManager.LoadLevel("Level 4");
        }

        // Update the player's animations based on their movement
        UpdateAnimations();
    }

    // Function to update the player's animations based on their movement and state
    private void UpdateAnimations()
    {
        AnimationState state;

        if (dirX > 0f)
        {
            state = AnimationState.running;
            transform.rotation = Quaternion.Euler(0f, 0f, 0f);
        }
        else if (dirX < 0f)
        {
            state = AnimationState.running;
            transform.rotation = Quaternion.Euler(0f, 180f, 0f);
        }
        else
        {
            state = AnimationState.idle;

        }

        // Set the animation state based on the player's vertical movement
        if (rb.velocity.y > .1f)
        {
            state = AnimationState.jumping;
        }
        else if (rb.velocity.y < -.1f)
        {
            state = AnimationState.falling;
        }

        // Update the animation state in the Animator component
        anim.SetInteger("state", (int)state);
    }

    // Function to check if the player is on the ground
    private bool onGround()
    {
        // Use a BoxCast to check for ground collision, returns true if on ground
        return Physics2D.BoxCast(collision.bounds.center, collision.bounds.size, 0f,
Vector2.down, .1f, onGroundMask);
    }
}
```

PlayerVisibility.cs (1.20)

```
/*
 * PlayerVisibility.cs
 * Author: William Dibble
 * Date: 24-04-2023
 *
 * Stores and retrieves the visibility status of the player.
 *
 */

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PlayerVisibility : MonoBehaviour
{
    public static bool IsInvisible { get; private set; }

    // Set invisiblity status of the player
    public static void SetInvisible(bool invisible)
    {
        IsInvisible = invisible;
    }

    // Get invisiblity status of the player
    public static bool GetInvisible()
    {
        return IsInvisible;
    }
}
```

PointFollower.cs (1.21)

```
/*
 * PointFollower.cs
 * Author: William Dibble
 * Date: 24-04-2023
 *
 * The PointFollower script is used to make an object move between a series of points.
 * This is used for the blades, the moving platforms, and the invisible bats.
 */

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PointFollower : MonoBehaviour
{
    [SerializeField] private GameObject[] points; // Array of points to move between

    private int currentPointIndex = 0; // Current point index

    [SerializeField] private float speed = 2f; // Movement speed

    [SerializeField] private bool alwaysOn = false; // If true, the object always
moves regardless of player's distance

    // Reference to the player's transform
    [SerializeField] private Transform player;

    // Distance threshold for starting the movement if not always on
    [SerializeField] private float maxDistance = 15f;

    private void Update()
    {
        bool shouldMove = false;

        // Check conditions for the object to move
        if (gameObject.CompareTag("Platform"))
        {
            shouldMove = true;
        }
        else if (alwaysOn)
        {
            shouldMove = true;
        }
        else if (Vector2.Distance(player.position, transform.position) <= maxDistance)
        {
            shouldMove = true;
        }

        // If the object should move, update its position
        if (shouldMove)
        {
            // If the object has reached its current point, move to the next point in
the array
            if (Vector2.Distance(points[currentPointIndex].transform.position,
transform.position) < .1f)
            {
                currentPointIndex++;
                if (currentPointIndex >= points.Length)
                {
                    currentPointIndex = 0;
                }
            }

            // Move the object towards the current point
            transform.position = Vector2.MoveTowards(transform.position,
points[currentPointIndex].transform.position, Time.deltaTime * speed);
        }
    }
}
```

RollingEnemyAI.cs (1.22)

```
/*
 * RollingEnemyAI.cs
 * Author: William Dibble
 * Date: 24-04-2023
 *
 * This script controls the AI behavior of the rolling enemy in the game.
 * The enemy follows a path to move towards the player's position, using the A*
pathfinding algorithm implemented through the Pathfinding package.
 */

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Pathfinding;

public class RollingEnemyAI : MonoBehaviour
{
    public Transform target;

    public float speed = 200f;
    public float nextWaypointDistance = 3f;

    Path path;
    int currentWaypoint = 0;
    bool reachedEndOfPath = false;
    public int health = 100;

    Seeker seeker;
    Rigidbody2D rb;
    private SpriteRenderer sprite;
    private BoxCollider2D collision;
    private AudioManager audioManager;

    void Start()
    {
        // Get required components for AI navigation and audio
        seeker = GetComponent<Seeker>();
        rb = GetComponent<Rigidbody2D>();
        sprite = GetComponent<SpriteRenderer>();
        collision = GetComponent<BoxCollider2D>();
        audioManager = FindObjectOfType<AudioManager>();

        // Update the path periodically
        InvokeRepeating("UpdatePath", 0f, .5f);
    }

    void UpdatePath()
    {
        // Calculate distance to target and determine maximum allowed distance
        float distanceToTarget = Vector2.Distance(transform.position,
target.position);
        float maxDistance = PlayerVisibility.GetInvisible() ? 8f : 13f;

        // If the target is too far away, don't update the path
        if (distanceToTarget > maxDistance)
        {
            return;
        }

        // Request a new path if the seeker is not currently calculating one
        if (seeker.IsDone())
        {
            seeker.StartPath(rb.position, target.position, OnPathComplete);
        }
    }

    void OnPathComplete(Path p)
    {
        // Set the new path if it is valid
        if (!p.error)
        {
            path = p;
            currentWaypoint = 0;
        }
    }

    void FixedUpdate()
    {
        // If there's no path, don't do anything
        if (path == null)
        {
            return;
        }

        // Calculate distance to target and determine if the enemy should move towards
it
        float distanceToTarget = Vector2.Distance(transform.position,
target.position);

        if (distanceToTarget > 13f)
        {
            if (PlayerVisibility.GetInvisible() && distanceToTarget > 8f)
            {
                return;
            }
        }

        // Move the enemy along the path
        if (currentWaypoint >= path.vectorPath.Count)
        {
            reachedEndOfPath = true;
            return;
        }
        else
        {
            reachedEndOfPath = false;
        }

        // Calculate direction and force to apply to the enemy
        Vector2 direction = ((Vector2)path.vectorPath[currentWaypoint] -
rb.position).normalized;
```

```
        Vector2 force = speed * Time.deltaTime * direction;

        // Halve the force if the player is invisible
        if (PlayerVisibility.GetInvisible())
        {
            force = (speed / 2) * Time.deltaTime * direction;
        }

        rb.AddForce(force);

        // Move to the next waypoint if close enough
        float distance = Vector2.Distance(rb.position, 
path.vectorPath[currentWaypoint]);

        if (distance < nextWaypointDistance)
        {
            currentWaypoint++;
        }

        // Determine which way the enemy should face based on the target's x position
        if (transform.position.x > target.position.x)
        {
            sprite.flipX = true;
        }
        else
        {
            sprite.flipX = false;
        }
    }

    // Function to apply damage to the enemy
    public void TakeDamage(int damage)
    {
        health -= damage;

        // Play damage sound
        audioManager.PlayHit2Sound();

        // If the enemy's health reaches 0, call the Die function
        if (health <= 0)
        {
            Die();
        }
    }

    // Function to destroy the enemy when its health reaches 0
    void Die()
    {
        // Play death sound
        audioManager.PlayHit1Sound();
        // Destroy the enemy object
        Destroy(gameObject);
    }

    // Function to handle collisions with other objects using OnTriggerEnter2D
    void OnTriggerEnter2D(Collider2D other)
    {
        if (other.gameObject.CompareTag("Trap"))
        {
            // Ignore collisions between the enemy and the trap object
            Physics2D.IgnoreCollision(collision, other, true);
        }
    }

    // Function to handle when the enemy stops colliding with other objects using
OnTriggerExit2D
    void OnTriggerExit2D(Collider2D other)
    {
        if (other.gameObject.CompareTag("Trap"))
        {
            // Restore collisions between the enemy and the trap object
            Physics2D.IgnoreCollision(collision, other, false);
        }
    }

}
```

StartMenu.cs (1.23)

```
/*
 * StartMenu.cs
 * Author: William Dibble
 * Date: 24-04-2023
 *
 * This script defines the behavior of the start menu in the game, including loading
levels, resetting the time scale, and playing sounds.
 *
 */


using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class StartMenu : MonoBehaviour
{
    // Reference to the Timer script
    private Timer timer;

    // Reference to the LevelManager script
    private LevelManager levelManager;

    // Reference to the AudioManager script
    private AudioManager audioManager;

    private void Start()
    {
        // Find the Timer object and get its Timer script component
        timer = FindObjectOfType<Timer>();
        // Find the LevelManager object and get its LevelManager script component
        levelManager = FindObjectOfType<LevelManager>();
        // Find the AudioManager object and get its AudioManager script component
        audioManager = FindObjectOfType<AudioManager>();
    }

    public void StartGame()
    {
        // Load Level 1 when Start Game is clicked
        levelManager.LoadLevel("Level 1");
        // Play selection sounds
        audioManager.PlaySelect1Sound();
        audioManager.PlaySelect4Sound();
    }
    public void Tutorial()
    {
        // Load Level 0 (Tutorial) when Tutorial is clicked
        levelManager.LoadLevel("Level 0");
        // Play selection sounds
        audioManager.PlaySelect1Sound();
        audioManager.PlaySelect4Sound();
    }

    // Load the main menu scene
    public void MainMenu()
    {
        // Reset time scale to 1 before returning to the main menu
        Time.timeScale = 1f;

        // Destroy the Timer object if it exists
        if (timer != null)
        {
            Destroy(timer.gameObject);
        }

        // Load the main menu scene
        SceneManager.LoadScene("Start Scene");

        // Play selection sounds
        audioManager.PlaySelect1Sound();
        audioManager.PlaySelect4Sound();
    }

}
```

Timer.cs (1.24)

```
/*
 * Timer.cs
 * Author: William Dibble
 * Date: 24-04-2023
 *
 * This script defines a Timer class with functionality for timing gameplay and
managing bullet counts.
 * It also has functions for starting and stopping the timer, and getters and setters
for bullet counts and bullet level counts.
 * Additionally, it has functions for subscribing to and unsubscribing from the
SceneManager's sceneLoaded event, and a function for handling the event,
 * which starts the timer when the Level 1 scene is loaded and stops the timer when
the Final Score Screen scene is loaded.
 */

using UnityEngine;
using UnityEngine.SceneManagement;

public class Timer : MonoBehaviour
{
    public float elapsedTime = 0f;
    private bool isTiming = false;

    // Number of bullets for each type of bullet
    private int bullet1Count = 0;
    private int bullet2Count = 0;
    private int bullet3Count = 0;
```

```csharp
private int[] bulletLevel1 = { 0, 0, 0 };
private int[] bulletLevel2 = { 0, 0, 0 };
private int[] bulletLevel3 = { 0, 0, 0 };
private int[] bulletLevel4 = { 0, 0, 0 };

private void Awake()
{
    // Make this game object persistent across scenes
    DontDestroyOnLoad(gameObject);
    // Subscribe to the sceneLoaded event
    SceneManager.sceneLoaded += OnSceneLoaded;
}

private void OnDestroy()
{
    // Unsubscribe from the sceneLoaded event
    SceneManager.sceneLoaded -= OnSceneLoaded;
}

private void OnSceneLoaded(Scene scene, LoadSceneMode mode)
{
    // Check if the loaded scene is Level 1
    if (scene.name == "Level 1")
    {
        // Start the timer
        StartTimer();
    }
    // Check if the loaded scene is the Final Score Screen
    else if (scene.name == "Final Score Screen")
    {
        // Stop the timer (commented out in the original script)
        // StopTimer();
    }
}

private void Update()
{
    // If the timer is running, update the elapsed time
    if (isTiming)
    {
        elapsedTime += Time.deltaTime;
    }
}

public void StartTimer()
{
    // Enable the timer
    isTiming = true;
}

public void StopTimer()
{
    // Disable the timer
    isTiming = false;
}

public float GetTime()
{
    // Return the current elapsed time
    return elapsedTime;
}
```

```
// Bullet count getters and setters
public int GetBullet1Count()
{
    return bullet1Count;
}

public void SetBullet1Count(int count)
{
    bullet1Count = count;
}

public int GetBullet2Count()
{
    return bullet2Count;
}

public void SetBullet2Count(int count)
{
    bullet2Count = count;
}

public int GetBullet3Count()
{
    return bullet3Count;
}

public void SetBullet3Count(int count)
{
    bullet3Count = count;
}

// Bullet level count getters and setters
public int[] GetBulletLevel1Count()
{
    return bulletLevel1;
}

public void SetBulletLevel1Count(int[] count)
{
    bulletLevel1 = count;
}

public int[] GetBulletLevel2Count()
{
    return bulletLevel2;
}

public void SetBulletLevel2Count(int[] count)
{
    bulletLevel2 = count;
}

public int[] GetBulletLevel3Count()
{
    return bulletLevel3;
}

public void SetBulletLevel3Count(int[] count)
{
    bulletLevel3 = count;
}
```

```
    public int[] GetBulletLevel4Count()
    {
        return bulletLevel4;
    }

    public void SetBulletLevel4Count(int[] count)
    {
        bulletLevel4 = count;
    }
}
```

TimerUICounter.cs (1.25)

```
/*
 * TimerUICounter.cs
 * Author: William Dibble
 * Date: 24-04-2023
 *
 * This script updates the TextMeshProUGUI component of the TimerUI with the current
time value from the Timer script, formatted as a whole number.
 *
 */

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using TMPro;

public class TimerUICounter : MonoBehaviour
{
    // Declare a reference to the Timer script
    private Timer timer;

    // Start is called before the first frame update
    void Start()
    {
        // Find the Timer script in the scene and assign it to the timer variable
        timer = FindObjectOfType<Timer>();
    }

    // Update is called once per frame
    void Update()
    {
        // Get the TextMeshProUGUI component attached to the game object
        // Set its text to the current time value from the Timer script, formatted as
a whole number
        GetComponent<TextMeshProUGUI>().text = timer.GetTime().ToString("F0");
    }
}
```

WeaponSelection.cs (1.26)

```
/*
 * WeaponSelection.cs
 * Author: William Dibble
 * Date: 24-04-2023
 *
 * This script handles weapon selection, bullet counts, and weapon-related actions for
the player character.
 * It includes methods for spawning bullets, updating UI elements, and collecting
bullets.
 */

using UnityEngine;
using UnityEngine.UI;
using TMPro;

public class WeaponSelection : MonoBehaviour
{
    // Array of weapon sprites
    [SerializeField] private Sprite[] weaponSprites;

    // GameObject that contains the image of the currently selected bullet
    [SerializeField] private GameObject bulletBoxObject;

    // Index of the currently selected weapon
    public int currentWeaponIndex = 0;

    // Text elements displaying the number of bullets for each type of bullet
    [SerializeField] private TextMeshProUGUI bullet1CountText;
    [SerializeField] private TextMeshProUGUI bullet2CountText;
    [SerializeField] private TextMeshProUGUI bullet3CountText;

    // Image component of the bullet box object
    private Image bulletBoxImage;

    // Transform representing the position to spawn bullets
    [SerializeField] private Transform firePoint;

    // Prefabs for each type of bullet
    [SerializeField] private GameObject bullet1Prefab;
    [SerializeField] private GameObject bullet2Prefab;
    [SerializeField] private GameObject bullet3Prefab;

    // Reference to the Timer script
    private Timer timer;

    // Reference to the Audio Manager script
    private AudioManager audioManager;

    private void Start()
    {
        // Set the initial weapon index to 0 (corresponding to the specific bullet)
        currentWeaponIndex = 0;

        // Get the Image component of the BulletBoxIMG object
        bulletBoxImage = bulletBoxObject.GetComponent<Image>();

        // Find the Timer object and get its Timer script component
        timer = FindObjectOfType<Timer>();

        // Find the Audio object and get its Audio Manager script component
        audioManager = FindObjectOfType<AudioManager>();
    }

    private void Update()
    {
        // Update the bullet count text for each type of bullet
        UpdateBullet1CountText();
        UpdateBullet2CountText();
        UpdateBullet3CountText();

        if (Input.GetKeyDown(KeyCode.V))
        {
            // Increment the current weapon index and wrap it around to 0 if it
exceeds the array length
            currentWeaponIndex = (currentWeaponIndex + 1) % weaponSprites.Length;

            // Update the weapon image
            bulletBoxImage.sprite = weaponSprites[currentWeaponIndex];
        }

        if (Input.GetKeyDown(KeyCode.C))
        {
            // Check if the current weapon index is 0 (corresponding to the specific
bullet)
            if (currentWeaponIndex == 0)
            {
                int bullet1Count = timer.GetBullet1Count();
                if (bullet1Count > 0)
                {
                    // Decrease the value of the bullet1Count variable and spawn a
bullet1Prefab at the fire point
                    timer.SetBullet1Count(bullet1Count - 1);
                    Instantiate(bullet1Prefab, firePoint.position,
firePoint.rotation);
                    audioManager.PlayFire1();
                }
            }
            // Check if the current weapon index is 1 (corresponding to the second
bullet)
            else if (currentWeaponIndex == 1)
            {
                int bullet2Count = timer.GetBullet2Count();
                if (bullet2Count > 0)
                {
                    // Decrease the value of the bullet2Count variable and spawn a
bullet2Prefab at the fire point
                    timer.SetBullet2Count(bullet2Count - 1);
                    Instantiate(bullet2Prefab, firePoint.position,
firePoint.rotation);
                    audioManager.PlayFire1();
                }
            }
            // Check if the current weapon index is 2 (corresponding to the third
bullet)
            else if (currentWeaponIndex == 2)
            {
                int bullet3Count = timer.GetBullet3Count();
                if (bullet3Count > 0)
                {
                    // Decrease the value of the bullet3Count variable and spawn a
bullet3Prefab at the fire point
                    timer.SetBullet3Count(bullet3Count - 1);
                    Instantiate(bullet3Prefab, firePoint.position,
firePoint.rotation);
                    audioManager.PlayFire1();
                }
            }
        }
    }
    // Method called when the object collides with a trigger collider
    private void OnTriggerEnter2D(Collider2D collision)
    {
        // Check if the collided object has the "Bullet1Collect" tag
        if (collision.gameObject.CompareTag("Bullet1Collect"))
        {
            int bullet1Count = timer.GetBullet1Count();
            // Check if the bullet1Count is less than the maximum value of 9
            if (bullet1Count < 9)
            {
                // Destroy the collided object, increase the bullet1Count and play the
collect sound effect
                Destroy(collision.gameObject);
                timer.SetBullet1Count(bullet1Count + 1);
                audioManager.PlayCollectSound();
            }
        }
        // Check if the collided object has the "Bullet2Collect" tag
        else if (collision.gameObject.CompareTag("Bullet2Collect"))
        {
            int bullet2Count = timer.GetBullet2Count();
            // Check if the bullet2Count is less than the maximum value of 9
            if (bullet2Count < 9)
            {
                // Destroy the collided object, increase the bullet2Count and play the
collect sound effect
                Destroy(collision.gameObject);
                timer.SetBullet2Count(bullet2Count + 1);
                audioManager.PlayCollectSound();
            }
        }
        // Check if the collided object has the "Bullet3Collect" tag
        else if (collision.gameObject.CompareTag("Bullet3Collect"))
        {
            int bullet3Count = timer.GetBullet3Count();
            // Check if the bullet3Count is less than the maximum value of 9
            if (bullet3Count < 9)
            {
                // Destroy the collided object, increase the bullet3Count and play the
collect sound effect
                Destroy(collision.gameObject);
                timer.SetBullet3Count(bullet3Count + 1);
                audioManager.PlayCollectSound();
            }
        }
    }

    // Method to update the bullet1CountText element
    public void UpdateBullet1CountText()
    {
        bullet1CountText.text = timer.GetBullet1Count().ToString();
        // Change the color of the text to red if the bullet count is at the maximum
value of 9
        if (timer.GetBullet1Count() == 9)
        {
            bullet1CountText.color = Color.red;
        }
        else
        {
            bullet1CountText.color = Color.black;
        }
    }
    // Method to update the bullet2CountText element
    public void UpdateBullet2CountText()
    {
        bullet2CountText.text = timer.GetBullet2Count().ToString();
        // Change the color of the text to red if the bullet count is at the maximum
value of 9
        if (timer.GetBullet2Count() == 9)
        {
            bullet2CountText.color = Color.red;
        }
        else
        {
            bullet2CountText.color = Color.black;
        }
    }

    // Method to update the bullet3CountText element
    public void UpdateBullet3CountText()
    {
        bullet3CountText.text = timer.GetBullet3Count().ToString();
        // Change the color of the text to red if the bullet count is at the maximum
value of 9
        if (timer.GetBullet3Count() == 9)
        {
            bullet3CountText.color = Color.red;
        }
        else
        {
            bullet3CountText.color = Color.black;
        }
    }
}
```

TransitionImageController.cs (1.27)

```
/*
 * TransitionImageController.cs
 * Author: William Dibble
 * Date: 24-04-2023
 *
 * This script controls the fade-in and fade-out process of an image, by changing the
alpha (opacity) of its colour over time.
 *
 */

using System.Collections;
using UnityEngine;
using UnityEngine.UI;

public class TransitionImageController : MonoBehaviour
{
    public float transitionTime = 1f;
    private Image transitionImage;
    private Canvas canvas; // Reference to the Canvas component

    private void Awake()
    {
        transitionImage = GetComponent<Image>();
        canvas = GetComponentInParent<Canvas>(); // Get the Canvas component from the
parent GameObject
```

```
    }

    // Method to start the fade-in process (image becomes visible)
    public void FadeIn()
    {
        canvas.sortingOrder = 2; // Set the sorting order to 2 before fading in
        StartCoroutine(Fade(0, 1));
    }

    // Method to start the fade-out process (image becomes invisible)
    public void FadeOut()
    {
        StartCoroutine(Fade(1, 0, () => canvas.sortingOrder = -1)); // Set the sorting
order to -1 after fading out
    }

    // Coroutine to handle the fade process (either fade-in or fade-out)
    // startAlpha: The initial alpha value (opacity) of the image
    // endAlpha: The target alpha value (opacity) of the image
    // onComplete: Optional action to perform after the fade process is complete
    private IEnumerator Fade(float startAlpha, float endAlpha, System.Action
onComplete = null)
    {
        float progress = 0;

        while (progress < transitionTime)
        {
            progress += Time.deltaTime;
            float alpha = Mathf.Lerp(startAlpha, endAlpha, progress / transitionTime);
            transitionImage.color = new Color(0, 0, 0, alpha);
            yield return null;
        }

        onComplete?.Invoke(); // Execute the onComplete action, if provided
    }
}
```

Weapon.cs (1.28)

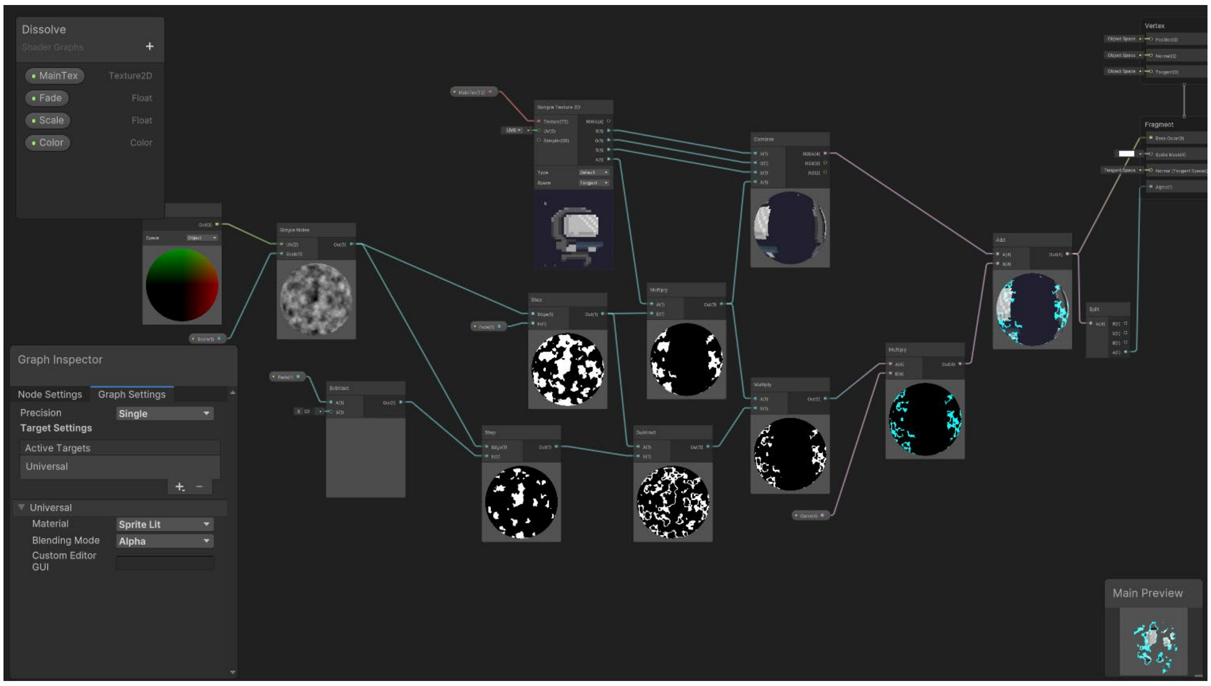
```
/*
 * Weapon.cs
 * Author: William Dibble
 * Date: 24-04-2023
 *
 * This script handles weapon firing logic and bullet counts for the player character.
 *
 */

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Weapon : MonoBehaviour
{
    [SerializeField] private Transform firePoint;
    [SerializeField] private GameObject bullet1Prefab;
    [SerializeField] private GameObject bullet2Prefab;
    [SerializeField] private GameObject bullet3Prefab;

    private WeaponSelection weaponSelection;
    // Reference to the Timer script
    private Timer timer;

    private void Start()
    {
        // Find the WeaponSelection script attached to the same game object
        weaponSelection = GetComponent<WeaponSelection>();
        // Find the Timer object and get its Timer script component
        timer = FindObjectOfType<Timer>();
    }

    private void Update()
    {
        if (Input.GetKeyDown(KeyCode.C))
        {
            // Check which weapon is currently selected
            switch (weaponSelection.currentWeaponIndex)
            {
                case 0:
                    // Check if the player has any bullet1 left
                    if (timer.GetBullet1Count() > 0)
                    {
                        // Instantiate the bullet1 prefab and decrease the bullet1
count
                        Instantiate(bullet1Prefab, firePoint.position,
firePoint.rotation);
                        timer.SetBullet1Count(timer.GetBullet1Count() - 1);
                    }
                    break;
                case 1:
                    // Check if the player has any bullet2 left
                    if (timer.GetBullet2Count() > 0)
                    {
                        // Instantiate the bullet2 prefab and decrease the bullet2
count
                        Instantiate(bullet2Prefab, firePoint.position,
firePoint.rotation);
                        timer.SetBullet2Count(timer.GetBullet2Count() - 1);
                    }
                    break;
                case 2:
                    // Check if the player has any bullet3 left
                    if (timer.GetBullet3Count() > 0)
                    {
                        // Instantiate the bullet3 prefab and decrease the bullet3
count
                        Instantiate(bullet3Prefab, firePoint.position,
firePoint.rotation);
                        timer.SetBullet3Count(timer.GetBullet3Count() - 1);
                    }
                    break;
            }
        }
    }
}
```

Dissolve.cs (1.29)

```
/*
 * Dissolve.cs
 * Author: William Dibble
 * Date: 24-04-2023
 *
 * This script is a component attached to a game object that provides the ability for
the
 * player to become invisible and visible again through a dissolve effect.
 *
 */

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Dissolve : MonoBehaviour
{
    Material material;

    bool isDissolving = false;
    float fade = 1f;
    private AudioManager audioManager;

    // Start is called before the first frame update
    void Start()
    {
        // Initialize the material variable with the material of the SpriteRenderer
component
        material = GetComponent<SpriteRenderer>().material;
        // Get the AudioManager component to play sounds later
        audioManager = FindObjectOfType<AudioManager>();
    }

    // Update is called once per frame
    void Update()
    {
        // Check if the "F" key has been pressed
        if (Input.GetKeyDown(KeyCode.F))
        {
            // If the player is not invisible, make them invisible and start the
dissolve process
            if (!PlayerVisibility.IsInvisible)
            {
                PlayerVisibility.SetInvisible(true);
                isDissolving = true;
                // Play the sound effect for becoming invisible
                audioManager.PlayInvisible1();
            }
            // If the player is already invisible, make them visible and reverse the
dissolve process
            else
            {
                PlayerVisibility.SetInvisible(false);
                isDissolving = false;
                // Play the sound effect for becoming visible
                audioManager.PlayInvisible2();
            }
        }

        // If the player is dissolving (becoming invisible)
        if (isDissolving)
        {
            // Decrease the fade value over time to make the player more transparent
            fade -= Time.deltaTime;

            // Make sure the fade value doesn't go below 0
            if (fade <= 0f)
            {
                fade = 0f;
            }

            // Set the "_Fade" property of the material to the new fade value
            material.SetFloat("_Fade", fade);
        }

        // If the player is not dissolving (becoming visible)
        else if (!isDissolving)
        {
            // Increase the fade value over time to make the player less transparent
            fade += Time.deltaTime;

            // Make sure the fade value doesn't go above 1
            if (fade >= 1f)
            {
                fade = 1f;
            }

            // Set the "_Fade" property of the material to the new fade value
            material.SetFloat("_Fade", fade);
        }
    }

    // Public method to access the isDissolving field
    public bool PlayerIsDissolving()
    {
        // Return the current state of the isDissolving field
        return isDissolving;
    }

}
```

Appendix G: Materials

Dissolve.mat



Dissolve
Shader Graphs

MainTex	Texture2D
Fade	Float
Scale	Float
Color	Color

Graph Inspector

Node Settings　Graph Settings

Precision: Single

Target Settings

Active Targets

Universal

Universal

Material: Sprite Lit

Blending Mode: Alpha

Custom Editor GUI