Dark Knight: An Insight in Player Performance through Camera Movement Mirroring

Rohan Yadav

2310103

A thesis submitted for the degree of Master of Science in Advanced Computer Science

Supervisor: Dr Richard Bartle
School of Computer Science and Electronic Engineering
University of Essex

August 2024

Abstract

This research explores the impact of camera movement on player performance and experience in a 2D side-scroller platformer. It investigates whether changing the camera orientation from the conventional left-to-right to a right-to-left perspective affects player behaviour, performance metrics, and perceived difficulty. Using Unreal Engine 5.3 (UE5), a 2D side-scroller was developed with two playtest modes, one featuring a normal left-to-right level progression and camera movement, and the other mirrors the experience. Experimental data on player performance, including completion time, success rates, and enemy defeats, were collected and analysed. The results showed no significant correlation between camera direction and overall player performance, suggesting that camera orientation does not substantially affect gameplay outcomes.

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Dr Richard Bartle, for his continuous support and guidance throughout the duration of this project.

I am also thankful to my friends and colleagues who not only provided constructive feedback but also played a crucial role as playtesters during the development process.

Table of Contents

List of Tables and Figures

Glossary

AAA: In the video game industry, AAA (Triple-A) is an informal classification used to classify video games produced and distributed by a mid-sized or major publisher, which typically have higher development and marketing budgets than other tiers of games.

Abilities: Special actions or powers available to characters in the game, often used in combat or to solve puzzles.

Character Balancing: The process of adjusting the abilities, strengths, and weaknesses of characters in a game to ensure fairness and competitive equilibrium.

Dash: A quick burst of movement that allows the player character to rapidly travel a short distance in the direction they are facing.

Frame Rate Drop: A significant decrease in the rate at which frames are rendered per second (FPS) in a video game or graphical application. Frame rate is a measure of how many unique consecutive images (frames) a game or application can produce in one second.

Indie Developer: An indie developer is a person who is working on its own projects without being backed by investors or venture capital.

Playtesters: Individuals who participate in the testing phase of a game, often before its official release.

Platformer: A genre of video games characterized by players controlling a character as they navigate environments that involve jumping between platforms, avoiding obstacles, and sometimes battling enemies.

Spammable: A term used in gaming to describe abilities, attacks, or actions that can be used repeatedly with little to no cooldown, cost, or consequence. Spammable abilities often allow players to continuously perform a certain action, such as firing a weapon, casting a spell, or executing a combo. The term can have both positive and negative connotations, depending on how balanced the mechanic is within the game.

Speedrunning: The practice of playing through a video game with the primary goal of completing it as quickly as possible.

Tanky: A term used in gaming to describe a character, unit, or hero with high durability, capable of absorbing large amounts of damage without being easily defeated.

Tilemaps: A method of designing game environments by arranging small, reusable images or "tiles" into a grid to form larger levels, backgrounds, or terrains. Each tile typically represents a small part of the environment, such as ground, walls, or other features. Tilemaps are commonly used in 2D games, particularly platformers, to create complex and varied environments efficiently.

Vertical-Slice: A fully functional portion of a game that represents a small cross-section of the final product, including all key gameplay mechanics, art, sound, and user interface elements.

1. Introduction

A lot of research has been done on the mechanics and design of video games, especially in relation to how they affect player performance and engagement. Among various game mechanics, player controls and camera perspective are important aspects that influence the game experience. Traditionally, 2D side-scrollers feature a standard left-to-right progression, where players start at the left side of the map and finish the level on the right side. This conventional design is deeply ingrained in the games industry, as evidenced by the many popular games that follow this pattern, such as *Super Mario Bros* [1], and *Sonic the Hedgehog* [2]. While this design is well-established, little attention has been given to exploring how different perspectives, specifically the direction of camera movement, could influence player performance.

1.1. Current Challenges

Despite the innovations in game mechanics, the impact of reversing traditional gameplay elements, such as camera perspective, remains underexplored. Not much research has been done on how changing the camera perspective—from the conventional left-to-right progression to a right-to-left one—affects player experience and performance. Understanding whether such changes in camera perspective can alter player experience and performance metrics, such as scores and completion times, is crucial for understanding game design.

1.2. Research Motivation

The goal of this study is to investigate the limits of conventional game design and comprehend how small adjustments made in isolation, such as camera mirroring, might affect player performance and behaviour. The purpose of this study is to determine whether the two different playtest modes for a 2D platformer game—one with the typical left-to-right camera perspective and the other with a flipped right-to-left perspective—have an impact on player experience and performance.

1.3. Significance of the Study

This study is significant because it provides empirical evidence on the effects of camera perspective on player performance; by analysing the differences in player scores and performance metrics across the two playtest modes, this research contributes to a deeper understanding of how game mechanics can be optimized to enhance player engagement and satisfaction.

Furthermore, this study aims to answer the research question: *Does the direction of camera movement and game flow—whether progressing from left-to-right or from right-to-left—significantly affect player performance, behaviour, and perceived difficulty, in a 2D platformer?* This question is critical for game designers seeking to explore new avenues for improving player experiences in platformer games and other genres.

2. Background

2.1. Technical Context

This section explores the core technical components that were integral in the development of the game, providing context on the specific tools and technologies used. The game was developed primarily using Unreal Engine 5 (UE5), which was chosen because of its reputation in the gaming industry and dependability in 2D and 3D game development, as well as the motivation to learn UE5.

Unreal Engine and Unity have been the two most prominent game engines being used in the industry. However, since the latest release of Unreal Engine 5, there has been a noticeable shift in popularity towards Unreal.
In recent surveys, this shift in favour of Unreal Engine has become evident:

> "The big two in games engines are Unity and Epic's Unreal. Unreal's share has gradually increased, according to the reports, from 45 percent in 2021 to 63 percent in the latest survey. Unity's share supposedly dropped to 18 percent in 2023, from 50 percent in 2022, but in 2024 is back up to 47 percent. Fickle developers or unreliable surveys? We would suggest the latter. But there is a consistent pattern showing Unreal winning share – especially in the biggest studios. According to the report, in the media and entertainment industry "teams overwhelmingly prefer Unreal Engine (51 percent) compared to Unity (16 percent)." (Anderson, 2024) [3]

Unreal Engine 5 (UE5)
Unreal Engine version 5.3, developed by Epic Games, served as the primary development environment for the game. This engine is widely known for its cutting-edge graphics capabilities and robust toolset, making it a preferred choice for both indie and AAA developers.

In this project, Unreal Engine 5.3 was chosen for several key reasons:
- *Motivation to Learn*: A significant motivation behind this project was to gain experience with UE5, particularly in context of game development.
- *Industry Popularity*
- *Blueprint Systems*: One of UE5's standout features is its Blueprints Visual Scripting system, which allows developers to create complex gameplay mechanics without writing C++ code. This feature made it easier and faster to prototype and implement game logic during the development process.
- *Community and Documentation*: Unreal Engine benefits from an active community of developers and extensive official documentation, which provided a support for troubleshooting and enhancing game mechanics.

PaperZD
One of the essential tools utilized in this project was PaperZD, a plugin designed to enhance Unreal Engine's 2D capabilities, particularly for 2D sprite animation and management. PaperZD streamlines the animation workflow by integrating 2D animation *State Machines* (more on section 3.2.5. Game Mechanics Implementation). By using PaperZD, I was able to create smoother, more dynamic sprite animations for the game's main character and enemies. It offered a more user-friendly and powerful alternative to the native Paper2D system in Unreal, which has been less frequently updated.

Tiled Map Editor
Tiled is an open-source, tile-based map editor used to design and create the maps for the game. This tool was crucial for the game as it allowed faster level creation and tile editing.

Python
Python was employed post-development for the analysis of collected experiment data. My personal comfort with python, combined with its rich libraries for data manipulation and visualization, made it the ideal tool for this phase.
Following were the libraries used:
- *Pandas*: Utilized for data handling and manipulation.
- *Matplotlib*: Utilized for graphical visualizations.
- *Seaborn*: Used for advanced statistical visualizations, providing a high-level interface for drawing statistical graphics.
- *SciPy*: Applied for performing statistical analyses, including ANOVA and multi-factor ANOVA analysis.

2.2. Related Work

2.2.1. Movement Fluidity and Design Reflection in "*Celeste*"[4] by Maddy Makes Games

During the conceptualization of the game, a lot of inspiration was taken from *Celeste*, a 2D platformer, as similarly mentioned in the *Proposal* of this project. *Celeste* is renowned for its exceptionally fluid and responsive movement mechanics, which have been praised for their ability to create a seamless and engaging player experience. One of the initial goals of the project was to implement a fluid movement system, as displayed in *Celeste*, and further explained in a YouTube video titled "Why Does Celeste Feel So Good to Play"[5] by Game Maker's Toolkit.

I aimed to create a system that closely mirrors the responsiveness and precision seen in *Celeste*. This involved careful attention to character controls, ensuring that players feel a direct and intuitive connection to their in-game actions.

2.2.2 Speedrunning Inspiration and Level Design

Speedrunning, the practice of completing a game or a level in the shortest possible time, is a fundamental aspect of this game's design. The inspiration for incorporating speedrunning elements came from a YouTube video titled "How I made 10 hours of Gameplay with 30 seconds of content"[6] by I/O. This video explores how even a seemingly simple level can offer deep and engaging gameplay by allowing players to discover new paths and strategies as they become more familiar with the character's movement and mechanics.

In my game, I aimed to emulate this design philosophy by creating levels that reward players for mastering the game's controls and mechanics. Players can find quicker ways and improve their performance as they get more skilled, making the game more repayable.

2.2.3. Dash Mechanic Inspired by *Hades*

The dash mechanic in this game was heavily inspired by the action-packed gameplay of Hades,[7] developed by Supergiant Games. Hades is well known for his agile and responsive dash ability, which is essential for both fighting and movement and enables players to move swiftly past opponents and precisely navigate the landscape.

The implementation of the dash in my game was directly influenced by a YouTube video titled "Recreating Hades Dash in Unreal Engine 5 | Games Dissected"[8] by Ali Elzoheiry, that delves into the technical aspects of how the dash was coded in Hades using Unreal Engine 5 (UE5). By applying the techniques demonstrated in the video, I was able to create a dash system that not only allows for quick bursts of movement but also integrates smoothly with the game's overall mechanics, providing players with a powerful and satisfying movement option that enhances both gameplay and player engagement.

3. Method

3.1. System Design

3.1.1. Game Architecture

The architecture of this 2D side-scroller game has been meticulously designed to investigate the impact of camera movement on player performance and experience. Built within Unreal Engine 5.3 (UE5), the game leverages modern development tools to create a controlled environment where key variables, namely camera behaviour and player interactions, can be systematically manipulated and studied.

4.1.1.1 Game World Structure

The game world is composed of two distinct levels, designed to offer different challenges of varying difficulty to the players. The inclusion of multiple levels serves an experimental purpose, reducing the potential for situational bias and allowing for a broader collection of data points. This design ensures that variations in player performance can be more accurately attributed to the experimental conditions, rather than the specific characteristics of any single level.

Each level resets upon completion, ensuring that each playtest begins with identical conditions and is played in isolation—which is essential for preserving the integrity of the experiments. The decision to design a resetting game world was driven by the need to eliminate carry-over effects from previous sessions, thereby allowing the research to isolate the impact of camera movement on player performance.

4.1.1.2 Camera System

The camera system is a core component of the study, designed to directly test the hypothesis that camera movement affects player performance and experience.

Playtest 1: The camera tracks the player smoothly as they move from left to right, a standard approach in 2D side scrollers. When the player moves left, the camera remains static, snapping back to the player's position only when they stop.
Playtest 2: The roles are reversed, with the camera following the player's movement to the left and remaining static when moving to the right.

This camera setup is important for the study, as it allows for a controlled comparison between two conditions, providing empirical data on how camera movement might influence factors such as level success and completion time.

4.1.1.3 Core Gameplay Systems

Player Controls: Players choose from three distinct characters, each with unique abilities, providing a varied test bed for analysing player strategies. The control scheme is designed to be highly responsive, enabling precise movement. This design choice ensures that any differences observed in performance are more likely attributable to the experimental manipulation of the camera system rather than inconsistencies in player controls.

Enemy AI: The enemy AI is designed using UE5's Behavior Trees, which are configured to chase and attack the player. The simplicity of the AI ensures that there is no unpredictable AI behaviour, which could confound the results.

Game Logic and State Management
The game's logic is structured to provide a consistent flow from start to finish, with clearly defined states for each phase of gameplay—initiation, action, and completion. Upon starting a level, the game initializes the player's chosen character, loads the necessary assets, and begins tracking key metrics such as time to completion and enemy defeats.
This design ensures that each play session is uniform, allowing for the controlled collection of data across multiple sessions and levels. The state management system is crucial for maintaining consistency in experimental conditions, thereby enhancing the reliability of the collected data.

3.1.2. Tools and Technologies

The development of this 2D platformer game, designed to explore the impact of camera movement on player performance, utilized a carefully selected set of tools and technologies. Each tool was chosen for its ability to support both the creation of the game and the rigorous demands of experimental research.

Unreal Engine 5 (UE5)
The main development platform used for this project was Unreal Engine 5, and hence was responsible for most of the game systems and components, few are listed as follows:

- **2D Game Development:** Although traditionally known for its 3D capabilities, UE5 provided comprehensive tools for 2D development, including support for tile-based maps, orthographic camera setups, and sprite management through PaperZD plugin. These features were essential for building the game's levels, implementing character controls, and ensuring smooth camera behaviour.
- **AI Behavior Trees:** Implemented to create basic enemy AI that engages the player through simple chase and attack behaviours.
- **Blueprints:** Utilized as the visual scripting system to develop gameplay mechanics, manage game logic, and control interactions without the need for C++ coding.
- **Camera System:** Configured an orthographic camera to facilitate the controlled testing of different camera orientations, critical for the experimental setup.
- **Character Management:** Employed UE5's character components to define and manage player characters, each with unique abilities tailored to the gameplay.

Assets
Several assets used in this game were sourced from itch.io
These included sprites for characters, enemies, drops and all UI elements, as well as map tilesets for level creation.

3.2. System Implementation

The implementation phase of this 2D platformer was carefully planned out to meet the development and experimental research goals. Every component of the game was designed to provide smooth gameplay experience and to make it easier to conduct controlled experiments to evaluate the effects of camera movement.

3.2.1. Development Environment Setup

The development environment was configured to support the specific needs of a 2D side-scrolling platformer, leveraging UE5 as the core platform. The setup involved a combination of industry-standard tools, customized configurations, and management tools to ensure a smooth development process.

Unreal Engine 5 Configuration

The engine's flexibility allowed for the integration of specialized plugins and the customization of settings to meet the demands of this project. To optimize UE5 for a 2D side-scrolling game, specific configurations were made:

- **PaperZD Plugin:** was integrated into the project to improve the development of 2D animations and sprite management. This plugin provided essential tools for managing animated sprites as well as animation states, which are crucial in a 2D platformer, allowing for smooth transitions and precise control over character movements.
- **File Helper Blueprint Library Plugin:** was integrated into the project so that experiment data could be saved locally to the player's device, and then retrieved for further analysis.
- **2D Side-Scroller Setup:** The project was specifically configured to support 2D side-scroller gameplay. This involved setting up an orthographic camera, customizing the rendering settings for 2D visuals, and tailoring the physics engine to handle 2D interactions effectively.

Operating System

The development was conducted on machines running Windows 11, chosen for its compatibility with UE5 and other development tools. Windows 11's stability and support for the latest hardware drivers ensured that the development process was smooth and free from OS-related interruptions, which could have potentially impacted the progress and consistency of the experiment.

3.2.2. Assets

Maps

Three custom maps are designed specifically for the game.
The maps were created using the Dark Series City Tileset in the Tiled map editor. The finalised maps were then imported into UE5 for gameplay integration.

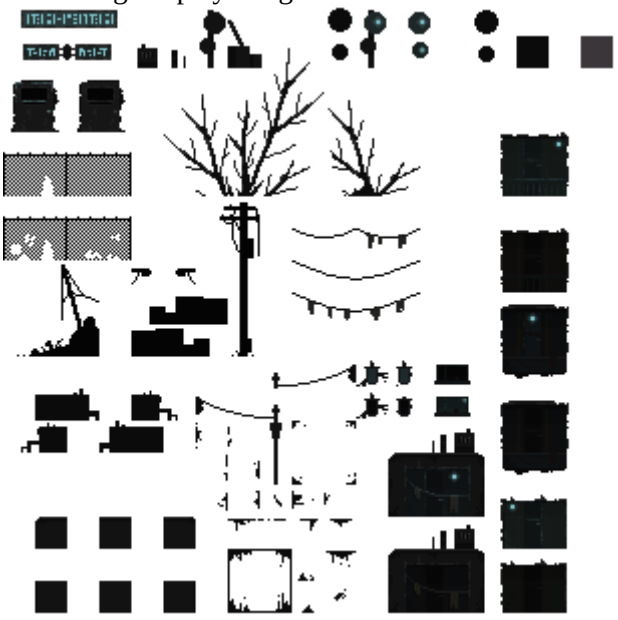

Figure 1: City Tileset by (Penusbmic, 2021)

Sprites

2D pixel art sprites are used for player characters and enemies.
Character sprites are taken from The DARK Series - Character Pack 1 by Penusbmic (itch.io) and The DARK Series - Character Pack 2 by Penusbmic (itch.io)

Enemy sprites are taken from The DARK Series - Character Pack 3 by Penusbmic (itch.io) and The DARK Series - Character Pack 4 by Penusbmic (itch.io)

UI
The User Interface is kept very minimal to ensure that player experience is not hindered by any distractions. The assets are taken from [The DARK Series - Animated Items & Blueprints by Penusbmic (itch.io)]

3.2.3. Game Flow Implementation

The game system is structured to facilitate data collection for the purpose of the experiment. The flow of the game is designed to guide the player through a series screens, eventually leading up to the core gameplay.

Selection Screens
Upon launching the game, the initial screen presents the option to select between two distinct Playtests (Playtest 1 or Playtest 2) as show in Figure 1: Playtest Selection Screen. This selection is crucial as each playtest is designed to decide the flow of the gameplay, which is either left-to-right or right-to-left.

Figure 2: Playtest Selection Screen

After a Playtest is selected, a *Hero Selection Screen* is presented to the player, allowing them to choose their character to play. During the conceptualization of this game, the characters planned were the Knight, Archer, and Mage. These characters were envisioned with specific set of abilities that players could enhance as they progressed through the game. However, during the actual development phase, adjustments were made to the character lineup. This change was driven by the availability of assets aligning with the theme of the game. The updated lineup includes the Assassin, Archer, and ShockSweeper (a tanky fighter), as shown in Figure 2: Hero Selection Screen.

Figure 3: Hero Selection Screen

Following the hero selection, the player is taken to a *Level Selection Screen* as shown in Figure 3: Level Selection Screen.

Figure 4: Level Selection Screen

Game Levels Implementation

In the context of game design, a level refers to a specific stage or segment of gameplay, typically designed to challenge the player in a progressive manner. Levels are curated experiences where game mechanics, difficulty and narrative elements like enemies are introduced in an isolated environment. On the other hand, a map is the physical representation of the layout of the game, which can consist of terrain, platforms and other interactive elements that the player can navigate through.

The design and implementation of game levels was crucial in showcasing the vertical slice of the game, providing both a comprehensive representation of gameplay mechanics as well as a controlled sandbox environment for data collection.

Given the scope of the project, a tutorial level and two distinct playable levels were developed:

- Tutorial: Teaching the players basic controls of the game and enemies to be encountered.
- City House: An easy level, designed to introduce the players to game's core mechanics. The map layout scales mostly horizontally, featuring only one out-of-map fall and two enemies. The map of the level is showcased in Figure 4: City House Map (Easy Level).
- City Rise: A hard level, designed to challenge the players. The map layout scales diagonally upwards, always maintaining the risk of out-of-map fall and featuring 6 enemies in total. The map of the level is showcased in Figure 5: City Rise Map (Hard Level).

Each level operates independently to ensure controlled data collection. Data monitoring begins as soon as the player starts the level and continues throughout the gameplay session. The collected data is stored when one of two conditions is met: either the player successfully clears the level by eliminating all enemies, or the player character is defeated.

Game Maps

While a map forms the structural backbone of a level, a single map can be reused across multiple levels. For this game, the maps were designed using the Tiled Map editor which allowed for quick prototyping and generation of 2D tilemaps. Additionally, Tiled was used to mirror the maps, creating variations that were employed in **Playtest 2**.

Figure 5: City House Map (Easy Level)

Figure 6: City Rise Map (Hard Level)

3.2.4. Camera System Implementation

The camera system in this 2D side-scrolling game was implemented using UE5's blueprint visual scripting system. This approach was chosen to ensure flexibility and ease of adjustment throughout the development process. To showcase the default view of the camera, *Figure 1: Screenshot taken within game*.

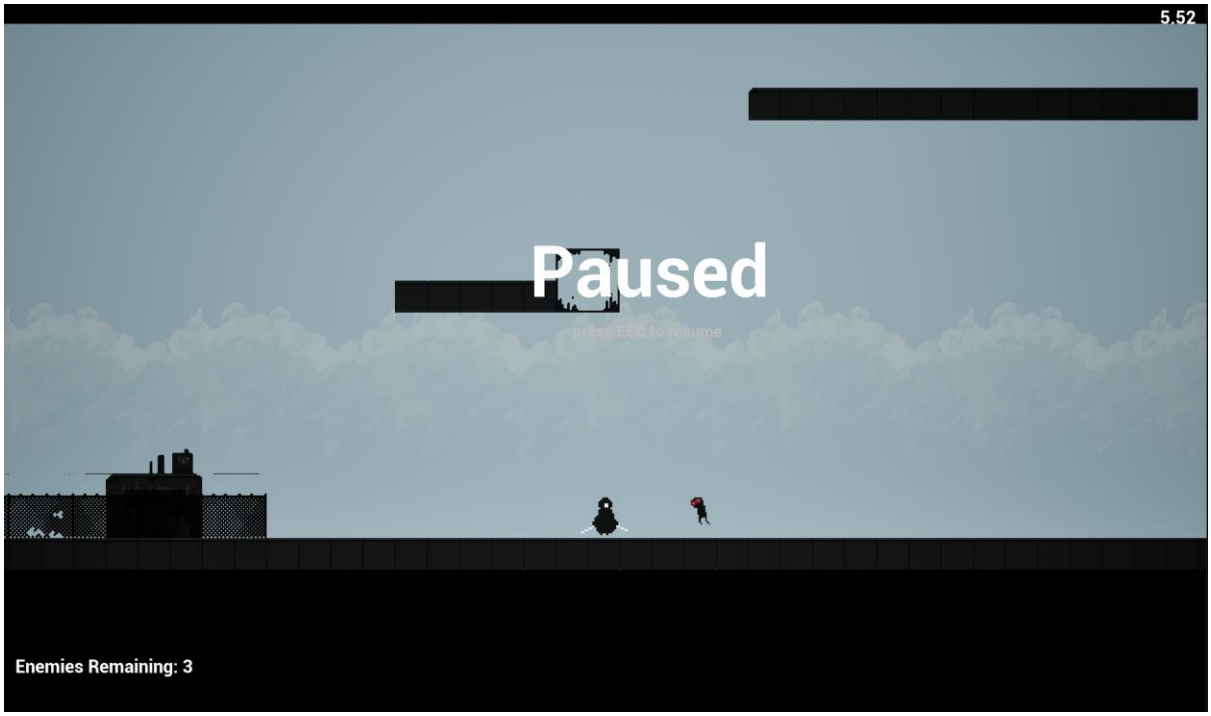

Figure 7: Screenshot taken within game

Camera Configuration

The camera system for the 2D side-scrolling game was configured with the following settings to achieve the desired visual and gameplay effects:

Setting	*Value*	*Description*
Projection Mode	Orthographic	Provides a consistent and distortion-free view suitable for 2D gameplay.
Otho Width	600	Defines the width of the viewable area.
Ortho Near Clip Plane	0	Configured to 0 units, ensuring that objects very close to the camera are rendered without clipping issues.
Ortho Far Clip Plane	10000	Extended to 10,000 units, ensuring that distant objects remain visible.
Aspect Ration	1.78	Maintained at 1.78, aligning with a standard widescreen format.

Table 1: Camera Configuration

Camera Movement Direction

The camera system's behaviour is contingent upon the playtest mode, which dictates the direction in which the camera follows the player. The camera's movement is governed by specific conditions as well as the playtest modes to achieve the desired experimental control and gameplay experience.

Playtest 1 (Left-to-Right): In this mode, the player starts from the left side of the map and the camera follows the player character's movement towards the right. This follows the conventional 2D side-

scrolling format, ensuring that the camera tracks the player's progression through the level from left to right.

Playtest 2 (Right-to-Left): Conversely, in this mode, the player starts from the right side of the map (as well as the map is flipped) and the camera follows the player's movement towards the left. This reversal tests the impact of a mirrored camera perspective on player performance and experience.

Camera Movement Constraints
The camera's behaviour regarding the opposite direction is constrained by specific conditions to prevent undesired or erratic camera movements:

- Camera does not follow the player during the opposite direction.
- Camera does follow the player based on the character states listed below:
    - Attacking
    - Jumping
    - Dashing

To address the scenario where the player moves out of the visible area of the game, the camera system is designed to snap back into the player's position. This ensures that the player can still follow back during the level, and the game stays playable until a desired end is reached.

3.2.5. Game Mechanics Implementation

Player Controls and Movement
The controls for the player were designed to be highly responsive, guaranteeing that any input from the player will be quickly reflected in the game. This responsiveness removed any potential control related issues that could have interfered with the experiment.

The player character is provided with following controls:

Action	*Key*	*Description*
Run	A (Left), D (Right)	The player character moves left or right at a constant speed. The running mechanic is straightforward, with no acceleration, deceleration, or additional mechanics. While running is not allowed during the attack animations.
Jump	Spacebar	The player character performs a standard jump with full air control, allowing precise movement in the air.
Dash	Left Shift	The player performs a quick burst in the direction they are facing upon a single press of the key. The dash has a limited number of uses and a cooldown period before it can be used again. The player is also invincible during the duration of the dash animation.
Light Attack	Left Mouse Button	The player executes a quick, spammable attack that can chain into a combo. This attack has a short range and is designed for rapid strikes against enemies.
Heavy Attack	Right Mouse Button	The player performs a powerful, spammable attack. While it deals more damage than the light attack, it is not part of a combo system and is typically slower.

Table 2: Player Input Controls

State Machines

The effectiveness of player controls and movement not only hinges on the responsiveness of the inputs, but also on the player animations; as they provide consistent feedback to player for different actions, allowing for better gameplay experience. To ensure that animations complemented the responsiveness of the controls, PaperZD plugin was integrated within UE5 for managing 2D animations.

A **State Machine** in PaperZD is a system that controls the flow of animations based on current "**State"** of the character. Each state represents a specific action or condition, such as **Idle**, **Walking**, **Jumping**, or **Attacking**, and dictates which animation should be played. Different states are connected to each other through **Transition Rules**, which are logical conditions that allow transitioning from one state to another, often connected to player input or an event.

Locomotion State Machine*:* This state machine is responsible for managing the movement states of the player character, namely *Idle, Walk, Jump, Fall, Hit, Death,* and *Dash.*

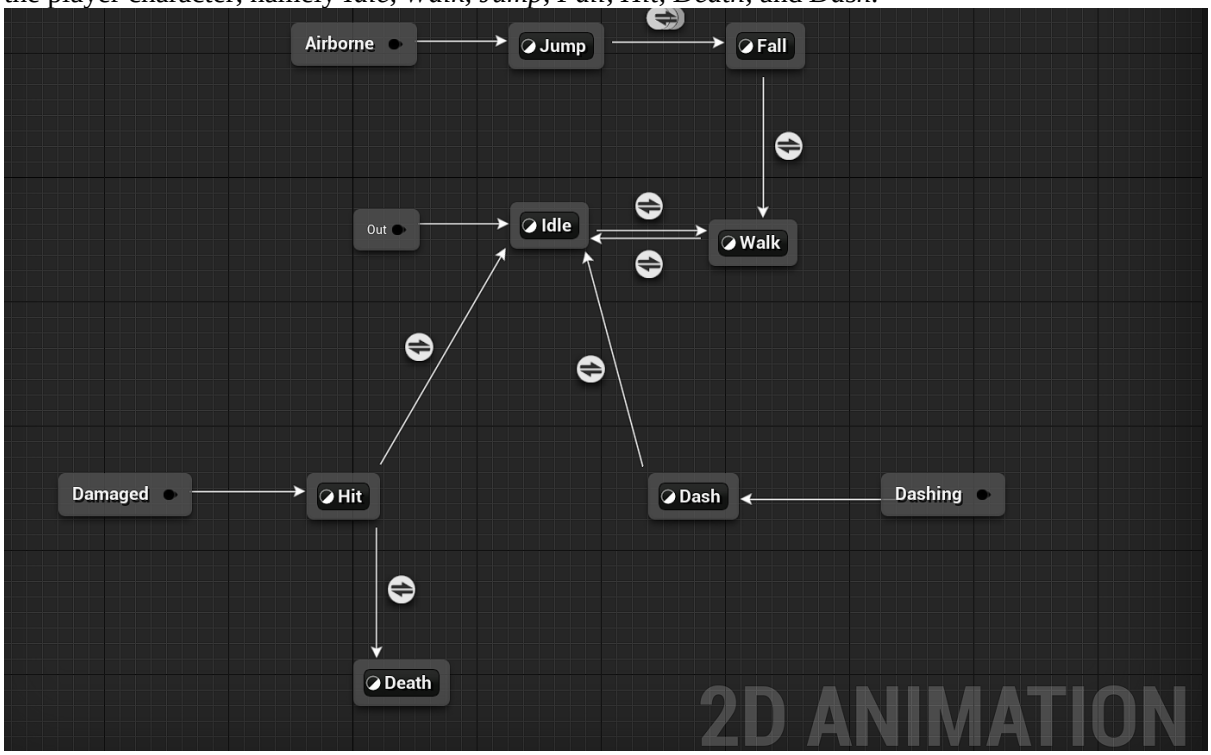

Figure 8: Locomotion State Machine for Player

Combat State Machine*:* This state machine is responsible for managing the attack states and their following chain attacks.

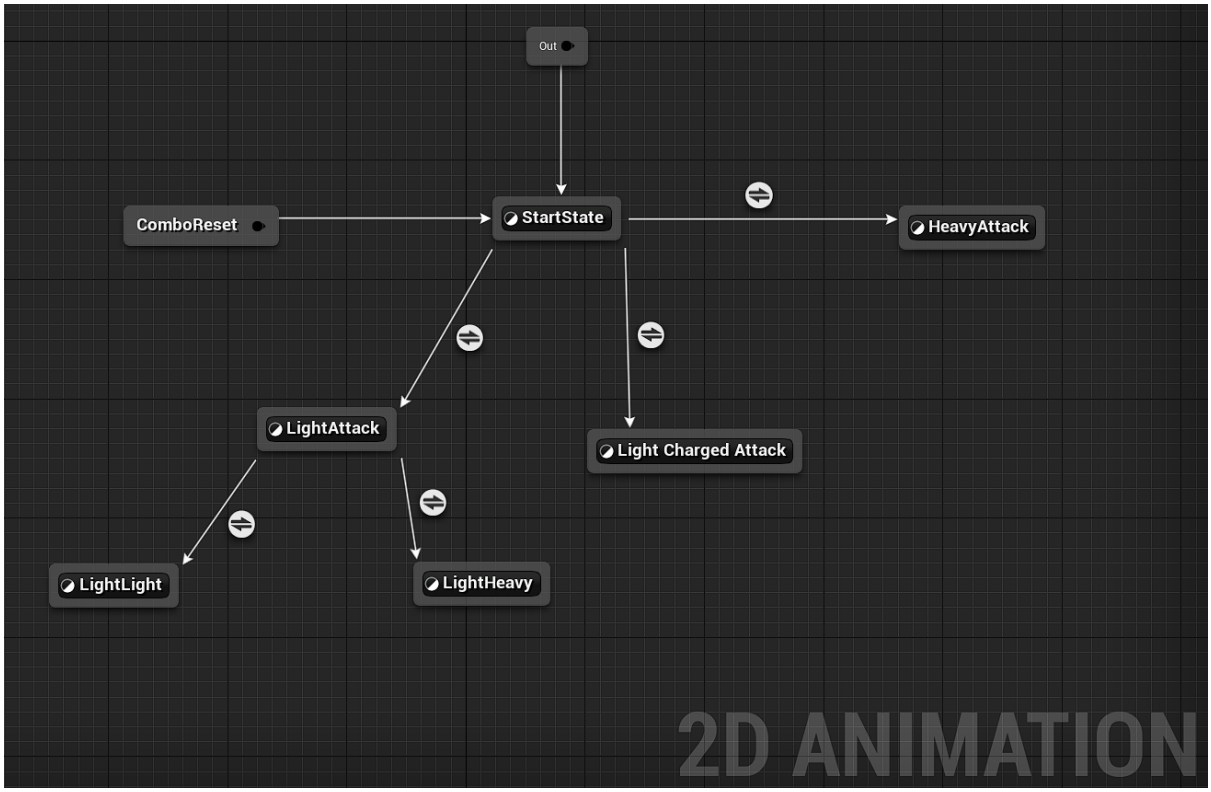

Figure 9: Combat State Machine for Player

Enemy AI

Behaviour Trees from UE5 were used to implement the enemy AI, as seen in Figure 11: Enemy AI Behaviour Tree in UE5.

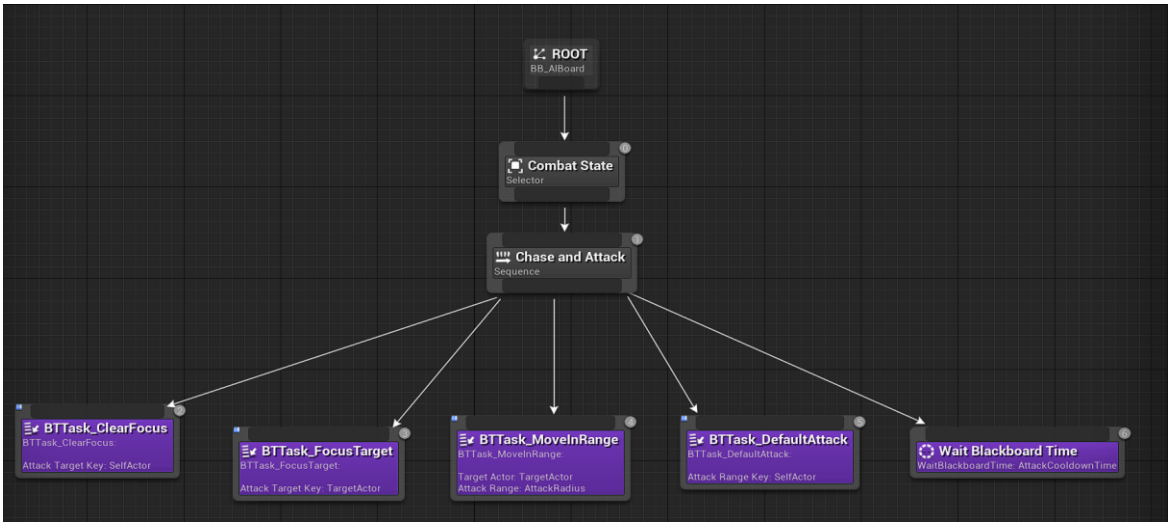

Figure 10: Enemy AI Behaviour Tree in UE5

The Behaviour Tree organizes the AI's decision-making process through a series of tasks, which are executed from left to right. Once the AI successfully completes a task, only then it moves on to the next task. On any enemy interruption, either through player or world interaction, the behaviour tree resets and starts executing the first task.

Below is the description of each task performed by the enemy AI.

Task Name	*Description*

BTTask_ClearFocus	Clears the AI's current target focus, resetting its attention to avoid confusion.
BTTask_FocusTarget	Re-focuses the AI's attention on the player, ensuring it targets the correct actor.
BTTask_MoveInRange	Moves the AI character within the attack range of the player, positioning it for combat.
BTTask_DefaultAttack	Executes the enemy's standard attack on the player once within range.
Wait Blackboard Time	Pauses the enemy's actions, based on the attack cooldown time, before the next attack cycle.

Table 3: Behavior Tree Tasks for Enemy AI

3.3. Experiment Design

The primary objective of this experiment is to evaluate how different camera movement affects player performance. Two playtest modes are designed to showcase the different camera behaviour, as mentioned in detail in section *3.2.3. Camera System Implementation*. The goal is to identify any measurable differences in player performance metrices.

Variables
Independent Variables:
- Camera perspective (based on the selected playtest), either left-to-right or right-to-left
- Choice of character
- Choice of Level

Dependent Variables: Player performance metrices.
Controlled Variables: Game mechanics, level design, enemy AI, player controls and environment.

In this experiment, the choice of character (Assassin, Archer, or ShockSweeper) and the level difficulty are treated as *Independent Variables*. By treating these factors as independent variables, we aim to capture a broader range of data that will allow for comparisons across different character types and levels of difficulty.

This approach provides insight into how different character playstyles and level difficulty, might affect the play performance metrices across different *Playtests*. By incorporating these variables, we ensure that the data collected reflects a wider range of gameplay scenarios, offering a more comprehensive understanding towards the effects of camera movement.

Hypothesis
Players will show no statistical correlation across all performance metrics, when comparing both playtest modes.

Participants and Procedure
Participants will be recruited from a diverse pool of players with varying levels of gaming experience, ranging from novices to experienced gamers. This approach allows the study to capture data from a broader audience, representing different skill levels. The varying experience levels are not a concern since the experiment focuses on comparing individual player performance across different playtests. A minimum of 15 participants will be recruited to ensure sufficient data collection for further statistical analysis.

Tracked Metrices
To evaluate the impact of camera direction, the following metrics will be recorded for each playtest session:
- **Character Selected**
- **Level Selected**

- **Level Success Rate:** Whether the player successfully completed the level, or the player character died.
- **Level Completion Time:** Time taken to finish the level or for player character to die.
- **Enemies Defeated:** The number of enemies eliminated by the player during the level.

3.4. Experiment Implementation

The experiment was implemented through a custom-built 2D side-scrolling game developed using UE5. The game was carefully structured to ensure that each playtest provided a controlled environment for the accurate collection of data.

Experiment Setup
The experiment is hosted on a standalone executable (.exe), allowing participants to engage with the game in a reliable and consistent manner. Upon launching the game, players are presented with a selection screen to choose between the two playtest modes: **Playtest 1** (left-to-right) and **Playtest 2** (right-to-left). Both playtests feature identical mechanics, levels, and challenges, and the variations are mentioned in Table 2: Playtest Configuration.

	Playtest 1	*Playtest 2*
Camera Movement while player is moving Right?	Yes	No
Camera Movement while player is moving Left?	No	Yes
Level Mirrored	No	Yes
Player Start Location	Left side of the map	Right side of the map

Table 2: Playtest Configuration

Data Collection and Storage
The game starts tracking player performance metrices as soon as the level starts. Once the level end is reached, either through completion or player character's death, then the tracked data is saved locally. The **File Helper Blueprint Library Plugin** was used to save the data locally on the player's device and then shared ahead for further analysis.

Test Environment
To ensure consistency across participants, the game is run on machines using **Windows 11** with standardized hardware configurations. This eliminates potential variations in performance due to hardware differences and ensures that gameplay remains smooth across all participants.

3.5. Testing

The testing phase was crucial for verifying the integrity and effectiveness of the game's design, implementation and the experimental setup. It ensures that the game functions as intended and that the

collected data accurately represents player performance under different *Playtests*. This section outlines the various aspects of the testing process.

During the development of this game, I undertook both unit testing and integration testing to ensure the stability and functionality of the game's systems. Although the testing methods were applied manually, they played an essential role in identifying and addressing issues that arose during the development process.

Unit Testing

Unit testing was conducted to verify the correctness of individual components within the game. This testing focused on the following areas:

- **Player Controls**: To guarantee precise and responsive execution, every control action—including movement, jumping, dashing, and attacks—was tested. Unit tests validated that player inputs were correctly mapped to in-game actions and that there were no unintended interactions between controls.
- **Camera System**: Tests were performed to ensure the camera's movement and behaviour adhered to the specified configurations for both *Playtest* modes. This included verifying that the camera followed the player appropriately in the left-to-right mode and correctly tracked movement in the right-to-left mode.

Integration Testing

Integration testing ensured that the various systems within the game interacted correctly and produced the expected outcomes:

- **Player and Camera Interaction**: Integration tests verified that the camera system interacted seamlessly with player controls. This involved checking that the camera followed the player correctly and that any edge cases, such as rapid direction changes, did not result in erratic camera behaviour.
- **Gameplay Mechanics**: The interaction between different gameplay elements, such as character abilities, enemy behaviour, and level design, was tested to ensure cohesive gameplay. This included scenarios where multiple mechanics interacted simultaneously, such as combat while moving or jumping.
- **Game Flow**: Integrated testing ensured that the game's flow remained cohesive and intentional as different systems interacted across various gameplay states. This includes verifying transitions between selection screens, maps and seamless state changes, such as shifting from active gameplay to menus.

Usability Testing

Usability testing was performed to ensure that the game was accessible and user-friendly. This included:

- **User Interface (UI)**: Ensured that the UI elements (e.g., selection screens, gameplay elements) were minimal and did not obscure the gameplay.
- **Instructions and Tutorial**: Verified that the tutorial level effectively taught players the basic controls and mechanics without causing confusion**.**
- **Map Size:** Through various iterations, the current map size was chosen, which is 200 tiles wide and 32 tiles high; each tile being a 16x16 pixel sprite. The small size of the map allowed for a faster level completion, important for a faster level clear and data collection.
- **Camera System:** An orthographic width of 600 was chosen for the camera to balance gameplay visibility and challenge. This width was designed to ensure that players could view enough of the level to follow the progression and make strategic decisions. However, this choice also introduced a risk of reduced awareness to potential falls out of the map.

Playtesting

Playtesting was conducted with the assistance of several participants to evaluate the overall gameplay experience and the effectiveness of the experimental setup. Both playtest modes were played by testers to get a variety of input on the user experience and gameplay. Through this process, all the

components of the game, as well as the experiment setup, were tested. These sessions were instrumental in refining the game features, level designs and bug fixes. The collective feedback from these early testers provided valuable guidance for a more polished game.

Bug Fixing
Bug fixing was an integral and ongoing part of the development process, with a continuous focus on addressing issues that could hinder the experiment's integrity. Throughout the development cycle, I prioritized fixing bugs that impacted gameplay mechanics, performance, and user experience. While I succeeded in fixing numerous critical issues, some minor bugs persist and might have influenced certain aspects of gameplay as well as the experiment setup.

3.6. Risks

Throughout the game's development process, several risks were identified that could potentially impact both gameplay experience and integrity of the experiment. Efforts were made to mitigate these risks to ensure the experiment remained valid and that the player experience was not significantly compromised.

Compatibility Issues
The game was developed using Unreal Engine 5.3 on a Windows 11 machine, with no testing conducted across other hardware configurations. This singular focus on one platform introduces the risk of compatibility issues when the game is run on different systems. Potential issues include:

- Game Failure to Start: The game may not launch or function as expected on systems differing from the development environment.
- Random Crashes: Unanticipated crashes could occur, leading to disrupted gameplay and potential loss of data.
- Inconsistent UI: Variations in user interface display might affect usability and player experience.

Bugs Affecting Gameplay
Despite rigorous manual testing and bug fixing, a few minor bugs remain within the game. Some of them are listed as follows:

- ShockSweeper sprites are not perfectly cantered, hence the hero animations seem bit misaligned, which can worsen the gameplay experience.
- Camera movement logic is not perfected, as there have been instances of glitches leading to player character endlessly running out of view.

Bugs such as graphical glitches or minor physics inconsistencies, especially with collision, may cause frustration and could influence the data.

Mitigation: Efforts were made to prioritise any bugs that could hinder the experiment's validity, and ongoing bug fixing was a constant part of the development process.

Camera and Visibility Issues
The orthographic camera width of 600 units was chosen to balance visibility with gameplay challenge. Risks include:

- Limited Visibility: Insufficient visibility could obscure critical gameplay elements, making it harder for players to navigate and impacting their ability to complete levels.
- Unintended Difficulty: The risk of increased difficulty due to limited visibility, coupled with camera glitches, could lead to player frustration and affect gameplay consistency.

Mitigation: Usability testing was conducted to ensure that the camera's orthographic width provided adequate visibility while keeping the intended level of challenge.

Performance Limitations
Potential performance issues such as frame rate drops, or input lag could affect the game's playability and the reliability of experimental data. This can result in an uneven gameplay experience, particularly on lower-end hardware.

Mitigation: The experiment was only conducted on systems which experienced no performance issues while running the game.

Playtester Behaviour and Data Bias
A significant risk in the experiment involves the way playtesters interacted with the game. If playtesters played the game in an overly organized or systematic manner—such as memorizing levels, optimizing movement paths, or rehearsing responses to game events—this could introduce bias into the data.

4. Results
This chapter presents the results of the experiment designed to evaluate the effect of camera movement (represented by the *Playtest* mode) on the player performance metrices. As mentioned previously, two versions of the game were tested: one where the camera follows a conventional left-to-right perspective (Playtest 1), and the other where the camera follows a reversed right-to-left perspective (Playtest 2).

For each playtest session, three performance metrics were recorded:
1. **Completion Time (in seconds)**: The time taken to complete each level.
2. **Number of Enemies Eliminated**: The total count of enemies eliminated by the player during the level.
3. **Success Rate (%)**: Whether the player successfully completed the level.

Around five hundred data points were gathered, across ten playtesters, during the experiment phase of this project. The data was further cleaned, and then statistical analysis was performed.

4.1. Descriptive Statistics
Before diving into inferential statistics, the basic statistical measures provide an overview of the data across different metrics.

Player Level Success Rate
Level success rates are shown across each independent variable (Playtest, Hero, and Level). Clear distinction can be seen across different playtests, as well as heroes and levels.

Playtest 1 seems to be tougher given the lower success rate, as compared to Playtest 2; but this can also be argued with new players unfamiliarity with the game and hence usually failing during first few rounds. Through personal observation of playtesters, I can confidently say that Playtest 1 is almost always the first selection on game launch, and hence could be the reason of bias.

	Level Fail	*Level Success*	*Success %*
Playtest 1	198	131	39.82
Playtest 2	102	93	47.69

Table 3: Level Success % per Playtest

The success rate across each hero is drastically different from each other. This might be an indicator of character balancing issue.

	Level Fail	*Level Success*	*Success %*
Archer	65	54	45.38
Assassin	211	99	31.94
ShockSweeper	24	71	74.74

Table 4: Level Success % per Hero

The City House level, designed to be easier than City Rise, does show a much higher success rate, likely hinting at the level's difficulty.

	Level Fail	*Level Success*	*Success %*
City House	26	97	78.86
City Rise	274	127	31.67

Table 5: Level Success % per Level

Level success rate across all independent variables are analysed as well. No new conclusion, which hasn't been covered before, is found from it.

Hero and Level	*Success % (Playtest 1)*	*Success % (Playtest 2)*
Archer – City House	78.26	75
Archer - City Rise	40	24.24
Assassin – City House	27.27	82.61
Assassin – City Rise	27.50	28.95
ShockSweeper – City House	83.87	92.59
ShockSweeper – City Rise	77.78	46.43

Table 6: Level Success % Pivot Table

Level Time to Complete

The correlation matrix shows a moderate correlation between *Level Time to Complete* and *Enemies Eliminated*, implying that players who take longer time to complete the level, are generally killing more enemies.

	Level Time to Complete	*Enemies Eliminated*
Level Time to Complete	1	0.569
Enemies Eliminated	0.569	1

Table 7: Correlation Matrix of Level Time to Complete & Enemies Eliminated

The Figure 11: Level Time to Complete metrics, gives various statistical metrics for *Level Time to Complete* across all independent variables. An important thing observed from this statistic is the high *count* for *Assassin* in *City Rise* level, and a drastically low count for *City House* level. This is a clear

discrepancy between the number playtests performed.

			count	min	mean	max	std
Play Test	Hero Name	Level Name					
Play Test 1	Archer	CityHouse	18	5.98	12.321111	31.38	7.720817
		CityRise	22	8.13	13.417727	40.04	7.137832
	Assassin	CityHouse	3	10.59	19.290000	31.53	10.909610
		CityRise	55	9.62	12.045455	19.65	2.209885
	ShockSweeper	CityHouse	26	8.13	12.409231	40.84	6.571316
		CityRise	7	15.41	23.430000	41.85	9.066798
Play Test 2	Archer	CityHouse	6	6.06	10.591667	21.39	5.850092
		CityRise	8	7.69	14.048750	33.00	8.424767
	Assassin	CityHouse	19	8.11	10.022632	19.69	2.607431
		CityRise	22	9.94	14.202727	28.25	4.319356
	ShockSweeper	CityHouse	25	8.08	12.864800	56.45	9.570176
		CityRise	13	11.49	21.638462	36.74	7.951760

Figure 11: Level Time to Complete metrics on Level Success

No noticeably huge differences are observed in the mean values of *Level Time to Complete* across both playtests. Assassin in the City House level does show an oddity in observation, as the time difference is about 9 seconds.

	Mean Time to Complete difference
Archer – City House	±1.73
Archer – City Rise	±0.63
Assassin – City House	±9.27
Assassin – City Rise	±2.15
ShockSweeper – City House	±0.45
ShockSweeper – City Rise	±1.79

Table 8: Mean Difference across playtest for Level Time to Complete

4.2. Inferential Statistics

4.2.1. Analysis of Variance (ANOVA)

To test whether the differences between the two *Playtest* modes towards the player performance metric, namely *Level Time to Complete*, an Analysis of Variance (ANOVA) was conducted. Three ANOVA results were recorded, one using the whole dataset, one with just the successful rounds, and the last one with just the unsuccessful rounds.

Hero Name	***Level Name***	***F-statistic***	***p-value***	***Significance***
Assassin	City Rise	5.275	0.0224	Significant
Assassin	City House	2.015	0.1655	Not Significant
ShockSweeper	City Rise	1.081	0.3055	Not Significant
Archer	City Rise	0.281	0.5972	Not Significant
Archer	City House	0.852	0.3636	Not Significant

ShockSweeper	City House	0.020	0.8894	Not Significant

Table 9: ANOVA result for dataset

Hero Name	***Level Name***	***F-statistic***	***p-value***	***Significance***
Assassin	City Rise	8.367	0.0050	Significant
Assassin	City House	12.348	0.0022	Significant
Archer	City Rise	0.042	0.8396	Not Significant
Archer	City House	0.250	0.6221	Not Significant
ShockSweeper	City Rise	0.210	0.6523	Not Significant
ShockSweeper	City House	0.040	0.8432	Not Significant

Table 10: ANOVA result for Level Success

Hero Name	***Level Name***	***F-statistic***	***p-value***	***Significance***
Assassin	City Rise	2.500	0.115	Not Significant
Assassin	City House	0.0004	0.9837	Not Significant
Archer	City Rise	0.455	0.5104	Not Significant
Archer	City House	0.028	0.8683	Not Significant
ShockSweeper	City Rise	0.563	0.4867	Not Significant
ShockSweeper	City House	1.066	0.3491	Not Significant

Table 11: ANOVA result for Level Failure

By analysing the ANOVA results we can see that apart from the *Assassin* character, no significant differences were found in the player performance (measured by *Level Time to Complete*).

4.2.2. Multifactor ANOVA

Source of Variation	***Sum of Squares*** *(sum_sq)*	***Degrees of Freedom*** *(df)*	***F-statistic*** *(F)*	***p-value*** *(PR(>F))*	***Interpretation***
Play Test	0.067450	1.0	0.001735	0.9668	Not Significant
Hero Name	1792.347670	2.0	23.047381	2.61e-10	Highly Significant
Level Name	12.587064	1.0	0.323708	0.5696	Not Significant
Play Test - Hero Name	103.336539	2.0	1.328780	0.2657	Not Significant
Play Test: Level Name	25.620292	1.0	0.658891	0.4173	Not Significant
Hero Name - Level Name	677.495138	2.0	8.711752	0.00019	Significant
Play Test - Hero Name - Level Name	137.719016	2.0	1.770897	0.1712	Not Significant
Residual	19908.596618	512.0			Residual variation (error term).

Interpretations

Direct effects of independent variables:

- *Play Test:* There is no significant effect of *Playtest* on dependent variable.
- *Hero Name:* The highly significant p-value (2.61e-10) indicates a strong effect of *Hero* on the dependent variable. This suggests that different heroes perform differently overall.
- *Level Name:* There is no significant effect of *Level* on dependent variable.

Indirect effects of independent variables:

- *Play Test - Hero Name:* The interaction between *Playtest* and *Hero* is not significant.
- *Play Test – Level Name:* The interaction between *Playtest* and *Level* is not significant.
- *Hero Name – Level Name:* This interaction is significant, a low p-value (0.00019), indicating that the *Hero* varies significantly depending on the *Level*.
- *Play Test – Hero Name – Level Name:* The three-way interaction is not significant.

4.2.3. Hypothesis Outcome

The initial hypothesis, during the conceptualisation of the experiment, predicts no statistical correlation between player performance metrics and *Playtests*. Although, through evaluation of results we can see that indeed no such correlation was found, but given the limitations of the collected dataset, this hypothesis isn't satisfactorily proven.

5. Analysis

Firstly, I would address the elephant in the room, which is the *Assassin* character. This character was observed to be overused, with the intention of speedrunning, during playtesting, and was a crowd favourite as well. Due to these reasons, I suspect that there is some unintentional bias in this data, and thus no further conclusions will be made w.r.t the data from this character.

5.1. Multifactor ANOVA

The data collected from the playtesting sessions were analysed using ANOVA to determine if there were significant differences in player performance metrics across the two *Playtests*. The ANOVA results presented in Tables 9, 10 & 11, reveal statistically insignificant differences between the two player conditions.

To further expand on these findings, Multifactor ANOVA was also applied to the data, where the factors being the independent variables. The Multifactor ANOVA results presented in Table 12, also support the similar claim of no significant statistical correlation between the *Playtest* and Player Performance Metrices.

The results also indicate that there is a high correlation between the *Hero* and *Level Time to Complete*. Which seems correct, and likely attributed to character-specific traits. To further support this claim, Figure 12: Level Completion Time comparison across Playtests, shows obvious difference between each character w.r.t *Level Completion Time*.

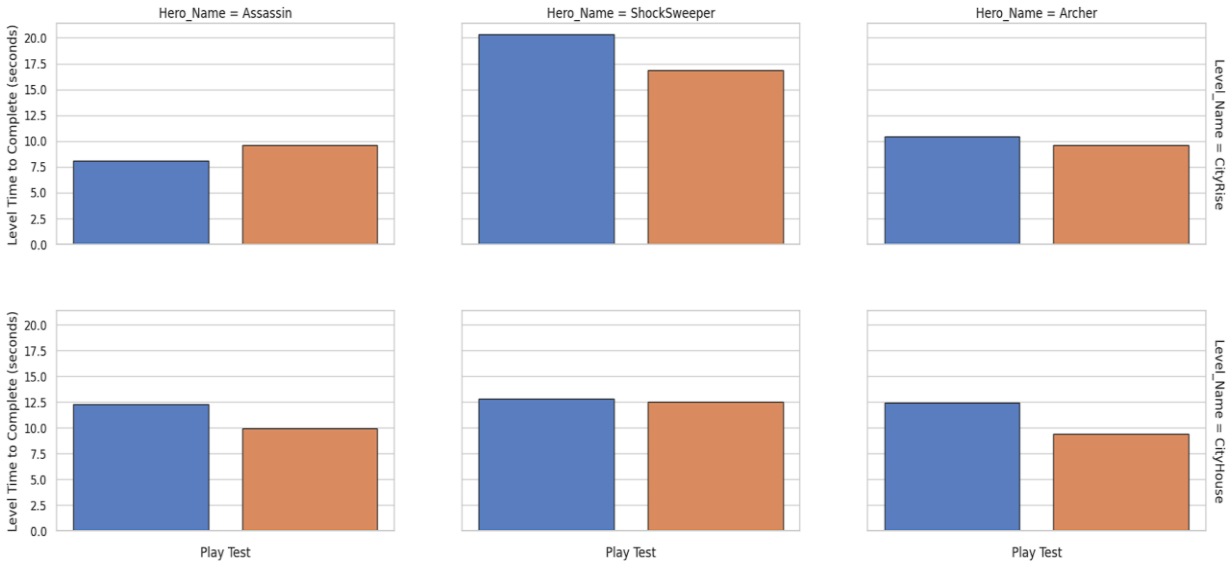

Figure 12: Level Completion Time comparison across Playtests

5.2. Graphical Analysis

To provide a clearer comparison of player performance across different heroes and *Playtests*, a normalized histogram was used to visualize the distribution of level completion times. The normalization ensures that the densities are comparable, regardless of the sample size for each *Hero*.

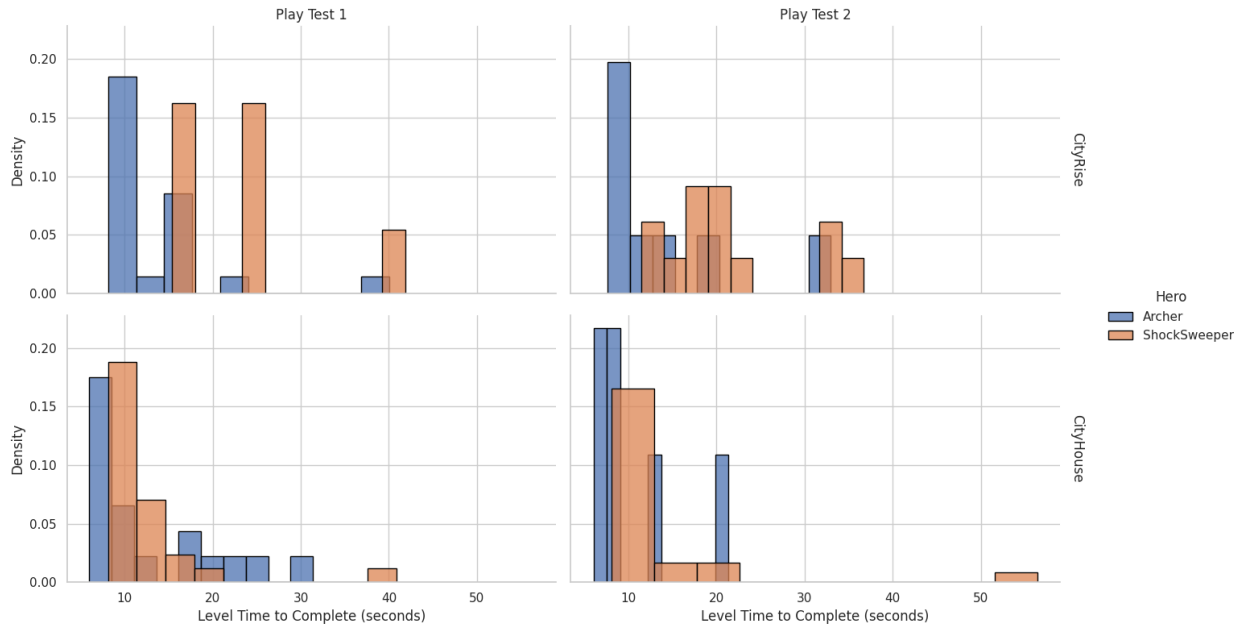

Figure 13: Normalized Histograms of Level Time to Complete by Playtest and Hero

From Figure 13 we can see that there is no significant difference in *Level Time to Complete* across *Playtests* for each *Level*.

5.3. Summary of Findings

- There's a strong suggestion of bias in data collected through *Assassin* character.

- No statistical correlation was found between the independent variables (*Playtest* and *Level*) and dependent variables (Player Performance Metrics).
- A strong relation was found between the *Hero* and *Level Time to Complete*.

6. Conclusion

6.1 Summary

This study explored the impact of camera movement on player performance in a 2D side-scroller platformer. The primary objective was to determine whether altering the conventional left-to-right camera perspective to a right-to-left orientation would significantly influence key performance metrics such as level completion time, enemy defeat rates, and success rates.

The findings revealed that, given the dataset, there was no statistically significant correlation between the direction of camera movement and player performance metrics across the different playtest modes. The Analysis of Variance (ANOVA) and Multifactor ANOVA results supported this conclusion, showing that while certain heroes exhibited differences in performance, these were more likely attributed to character-specific traits rather than the camera perspective itself.

Notably, the Assassin character's overuse by participants, driven by its appeal in speedrunning, introduced a potential bias in the data, making it difficult to draw definitive conclusions about the camera perspective's impact.

6.2. Future Work

Future efforts should focus on improving the game based on the findings of this study. Key areas for enhancement include:

- **Character Balancing:** Revising and balancing the abilities of all playable characters to ensure they offer equal challenges and opportunities, thereby reducing potential biases in player performance.
- **Camera System Refinement:** Fixing any glitches related to the camera movement, ensuring smooth transitions and consistent player visibility regardless of the direction of progression.
- **Gameplay Optimization:** Implementing tweaks to the game’s mechanics, such as adjusting difficulty levels and improving AI behaviour, to create a more engaging and seamless player experience.

Looking ahead, I am eager to continue developing and expanding this game beyond the scope of this dissertation. My goal is to address the identified issues and incorporate new features that will expand the game’s content and complexity. This includes adding more levels, introducing new characters with unique abilities, and much more!

References

[1] R. Nintendo and D4, "Super mario bros," *Game [NES].(13 September 1985). Nintendo, Kyoto, Japan*, 1985.

[2] Sega, "Sonic the Hedgehog [Video Game]," ed: Sega, 1991.

[3] T. Anderson. "Survey finds game engines used equally by non-game projects, rise of interest in open source Godot." Dev Class. https://devclass.com/2024/08/23/survey-finds-game-engines-used-equally-by-non-game-projects-rise-of-interest-in-open-source-godot/ (accessed 26 August, 2024).

[4] M. M. Games, "Celeste [Video Game]," ed: Maddy Makes Games, 2018.

[5] G. M. s. Toolkit, "Why Does Celeste Feel So Good to Play?," ed: https://www.youtube.com/watch?v=yorTG9at90g, 2019.

[6] I/O, "How I made 10 hours of Gameplay with 30 seconds of content – 2D Platformer Devlog 0," ed: https://www.youtube.com/watch?v=4U4iSuj4iX4, 2023.

[7] S. Games, "Hades [Video Game]," ed: Supergiant Games, 2020.

[8] A. Elzoheiry, "Recreating Hades Dash in Unreal Engine 5 | Games Dissected," ed: https://www.youtube.com/watch?v=newhrqTYAfs, 2024.

Appendix

Penusbmic - itch.io .

Tiled | Flexible level editor (mapeditor.org)

https://www.criticalfailure-studio.com/paperzd-documentation/

File Helper Blueprint Library in Code Plugins - UE Marketplace (unrealengine.com)

Link to CSEE Gitlab
23-24 CE901-CE911-CF981-SU / 23-24_CE901-CE911-CF981-SU_yadav_rohan · GitLab (essex.ac.uk)