Technical Design Document

Contents

Game Details

- **Game Name:** *The Fog*
- **Team Name:** *STAB Studios*

Team Members

List of technical team members and broad overview of their roles.

Name	Job Title	Responsibilities/Roles
Enter each team member		
Annelise	Artist	Creation and animation of zombie characters and other essential in-game assets. Collaborate with Chantelle to ensure a unified visual style between characters and environments. Work with Sion to ensure that art assets align with game mechanics and overall design.
Chantelle	Artist	Design and creation of environment art for the graveyard, swamp, and church settings. Develop environmental animations, lighting, and atmospheric effects to enhance the game's eerie ambiance. Coordinate with Annelise and Sion to ensure a seamless integration of environment art with game mechanics and character assets.
Brock	Programmer	Collaboratively work with Tom on gameplay programming, including the on-rails movement system, shooting mechanics, and local cooperative gameplay features. Share responsibilities in developing the game's user interface, including score

		tracking and display systems, as well as any necessary in-game menus. Work closely with Sion on implementing and fine-tuning game mechanics based on design inputs, ensuring a seamless integration of design ideas into the programming framework.
Tom	Programmer	Collaboratively work with Brock on systems programming tasks, such as optimizing the game for the arcade hardware, ensuring smooth performance and responsiveness. Share responsibilities in implementing gameplay features, and develop tools or scripts that may help streamline the development process. Maintain a clean, organized, and well-documented codebase to facilitate efficient collaboration and future adjustments, ensuring that both programmers can work
Sion	Designer	Designing game mechanics, level layouts, and player progression systems. Work closely with both artists and programmers to ensure the game design is well-implemented and aligns with the visual and technical aspects of the project. Conduct play-testing sessions, gather feedback, and iterate on the game design to enhance the player experience.

Game Concept

In “The Fog”, players traverse through distinct, spooky scenes, battling different types of zombies using an arcade gun control system and on rails movement. As they progress, they face increasing challenges that end

in a boss fight against the King Zombie. The game, with its scoring system and co-op mode, encourages competitive play and invites players to improve their skills and compete for high scores.

Technical Goals

What are the technical aspects of your game that your team aim to deliver? E.g. Challenging AI, Procedural Generated Levels, Interesting Jetpack mechanics.

Who will work on these goals?

Technical Goal 1 – Enemy State Machine Implementation

Who's Responsible: Brock

Description: Design a comprehensive State Machine to guide the AI of different enemy types across all game levels. The State Machine will dictate how enemies respond to player actions, environmental triggers, and other game events. Ensure that the behaviour nodes are modular for reusability across different enemy types and levels, and that they can dynamically react to varying game scenarios. Extensive testing should be carried out to ensure balanced and intended enemy states, with thorough documentation provided for team understanding and future adjustments.

Technical Goal 2 – Level Progression Mechanism

Who's Responsible: Brock

Description: Develop a robust mechanism to track player progress and score, enabling seamless transitions to subsequent levels upon current level completion. Ensure transitions between levels are smooth, retaining the player's score and other relevant game data. Conduct testing to identify and rectify any bugs or issues, and document the mechanism for team understanding and future adjustments.

Technical Goal 3 – Create an Effective Weapon Swap Mechanic

Who's Responsible: Tom

Description: Create a weapon swap mechanic that allows the player to swap to a fully automatic gun. The gun will operate on a timer, which can be represented as a slider, and will only become available when the slider reaches full value. The weapon slider can gradually decrease as the player uses the automatic gun. This will keep gameplay interesting and help the player when in desperate survival situations.

Technical Risks

What are the technical (i.e. related to programming) features and ideas most likely to cause problems? E.g A mechanic requires learning new design patterns and 3rd party libraries you've never used before. Or none of the programmers have experience with AI.

What can you do to reduce the risk? E.g Bob will perform additional research and will spend the first two days making a test project to prove this idea is possible. If not, we will cut the idea.

Put into dot points, too much text

Technical Risk 1 – Developing Diverse and Complex AI Behaviours

What's the risk about: The game "The Fog" features a variety of different enemy types, each requiring unique AI behaviours to challenge the player and keep the gameplay interesting. Even with some experience in using AI state machines, designing and implementing a wide range of distinct, smooth, and challenging behaviours for each enemy type is a complex task. The risk is that the AI may not behave as intended, could have performance issues, or may not provide a sufficient challenge or variety to keep players engaged.

How will risk be mitigated:

To address this challenge, the team will start by clearly defining the desired behaviours for each enemy type, breaking them down into manageable tasks. We will develop one AI behaviour at a time, ensuring that it is fully tested and refined before moving on to the next. Regular playtesting sessions will be conducted to assess the AI's performance and challenge level, gathering feedback to make necessary adjustments. This approach will facilitate the creation of efficient and varied AI behaviours, significantly enhancing the gameplay experience. If an effective state machine cannot be built. We will use a free pre-built Unity asset packs that will allow us to control enemies.

Technical Risk 2 – Implementing Local Co-op Features

What's the risk about: Implementing local co-op features requires a robust system for handling multiple players on the same screen, ensuring that the game provides a balanced and enjoyable experience for all players. This includes managing player input for multiple controllers or input devices, ensuring that the UI correctly displays information for all players, and balancing the gameplay to accommodate the increased player presence.

How will risk be mitigated:

To address this challenge, Brock and Tom will undertake comprehensive research on best practices for implementing local co-op in Unity. A simple prototype that includes basic co-op functionality will be developed to solidify this understanding and identify any potential challenges early on. This prototype will feature character movement, shooting, and displaying player-specific information on the UI. Once the prototype is successfully implemented and the team feels confident in their approach, the co-op features will be integrated into the main game. We will conduct regular playtesting sessions with multiple players to ensure that the co-op experience is enjoyable and balanced, making adjustments as necessary based on player feedback. If coop functionality cannot be achieved, we have a contingency plan to revert back to single player functionality.

Features/Mechanics/Tasks

A list of exactly what systems exist in your game and who is responsible, including scheduled dates for completion.

Feature/Mechanic	Who's responsible	Scheduled Date
Enter each feature/mechanic		
Player Movement	Brock/Tom	November 6th

Shooting/Aiming	Brock/Tom	November 9th
Reloading	Brock/Tom	November 9th
Enemy and environment assets	Annelise/Chantelle	November 9th
Damage/Health	Brock/Tom	November 10th-12th
Death	Brock/Tom	November 10th-12th
Co-op features	Brock/Tom	November 12th
UI / Scoring	Brock/Tom/Sion	November 15th
Enemy AI/Animations	Brock/Tom	November 19th
Items/Powerups	Brock/Tom	November 24th

Deliverables

What will you deliver at the end of production?

Deliverable	Who's Responsible	Who's the Owner
Enter each platform/input		
Windows Executable Build	Brock/Tom	Modern Arcades
WebgGL / Itch.io Build	Brock/Tom	Modern Arcades
Full Game Documentation	STAB full team	Modern Arcades
Art and Asset Package	Annelise/Chantelle	Modern Arcades
Sound and Music Package	Sion/Brock	Modern Arcades
Source Code	Brock/Tom	Modern Arcades

System Requirements

What devices is your game targeting? What's the recommended hardware? Portrait or Landscape mode on mobile?

Target Device 1 – Arcade Cabinet

- **Recommended Hardware: CPU:** Intel i5 (7th Gen) or equivalent
- **GPU:** NVIDIA GeForce GTX 1050 or equivalent
- **RAM:** 8GB
- **Storage:** 10GB free space
- **Display:** Full HD Monitor (1920x1080 resolution)
- **Audio:** Integrated stereo speakers
- **Networking:** Ethernet port for possible future online scoreboards or updates
- **Platform Specific Requirements:**
- **Controls:** Arcade cabinet controls including gun and buttons

Add link to reference cabinet

Target Device 2 – PC

- **Recommended Hardware: CPU:** Intel i5 (7th Gen) or equivalent
- **GPU:** NVIDIA GeForce GTX 1050 or equivalent
- **RAM:** 8GB
- **Storage:** 10GB free space
- **Display:** Full HD Monitor (1920x1080 resolution)
- **Audio:** Integrated stereo speakers
- **Networking:** Ethernet port for possible future online scoreboards or updates
- **Platform Specific Requirements:**
- **Controls:** Mouse and Keyboard

Third Party Tools

- Unity Engine 2022.3.11f1
- Visual Studio Community Edition 2022
- Photoshop
- Z Brush
- Nomad Sculpt
- Substance Painter
- Adobe Audition/Audacity/Logic Pro
- Perforce
- DOTween (HOTween v2) by Demigiant

File Formats

Models:

.FBX: A binary file format that accommodates complex 3D models and animations. This format is universally recognized and ensures seamless integration of intricate model designs and animations.

Textures:

.**PNG:** A preferred image format for game textures, combining lossless compression with support for transparency, resulting in high-quality visuals.

.JPG: An image format suitable for textures where transparency is not required, providing a balance between quality and file size.

Sounds:

.WAV: A lossless audio file format chosen to deliver the highest quality sound, ensuring crisp and clear audio playback.

Other Assets:

.MAT: Files that store material data, including how 3D models are rendered and references to the necessary textures.

.ANIM: Files containing animation data, capturing the motion and transformation of game objects.

.PREFAB: Prefabricated files that store predefined game objects, complete with their associated components and settings, allowing for consistent and reusable elements within the game.

Scripts and Code:

.CS: Files containing the C# programming code that drives the game's functionality and logic.

Scene and Project Files:

.UNITY: Files that capture the configuration and arrangement of game objects within a particular scene.

.UNITYPACKAGE: Comprehensive package files that may encapsulate various game assets, from scenes and prefabs to scripts and textures, facilitating straightforward sharing and importation of game components.

By adhering to these file formats, we ensure that all game assets are stored and delivered in a manner that is both standardized and optimal for high-quality game performance. This consistency across different types of assets contributes to a smoother development process and easier asset management.

Coding Conventions

What coding conventions will your team use? Everyone on the team should use the same conventions.

Use an existing guideline as a template, rather than create one from scratch. One example can be found at https://csharpcodingguidelines.com which also includes a Visual Studio plugin to automatically analyse your codebase (The C# Guidelines Analyzer).

Our team will adhere to a standardized set of coding conventions to ensure consistency, readability, and maintainability across our codebase. We have decided to use the guidelines provided by the C# Coding Guidelines (https://csharpcodingguidelines.com) as a template for our conventions.

Key aspects of our coding conventions include:

Naming Conventions:

- Classes and Methods: Use PascalCase for class names and method names.
- Variables: Use camelCase for variable names, and ensure names are descriptive and clear.
- Constants: Use uppercase letters with underscores separating words.

Code Layout:

- Indentation: Use spaces instead of tabs with a standard indent size of 4.
- Braces: Use the Allman style, placing open braces on a new line.
- Line Length: Aim to keep lines under 120 characters long for readability.

Commenting and Documentation:

- XML Documentation: Use XML comments for all public classes, methods, and properties.
- Inline Comments: Use inline comments sparingly, and only when necessary to explain complex pieces of code.

Error Handling:

- Use Exceptions: Use exceptions for error handling rather than return codes.
- Try-Catch: Only catch exceptions that can be handled or logged appropriately, and avoid empty catch blocks.

Best Practices:

- Code Reusability: Aim for modular and reusable code where possible.
- Use Design Patterns: Apply appropriate design patterns to solve common problems in a clean and efficient manner.

Testing:

- Naming Tests: Name test methods clearly, describing what they test and the expected outcome.

By adhering to these coding conventions, our team aims to maintain a high standard of code quality throughout the development of our game. This will facilitate easier collaboration, debugging, and future maintenance of the codebase.

Source Control

Which source control will be used? What rules should all team members adhere to when using source control?

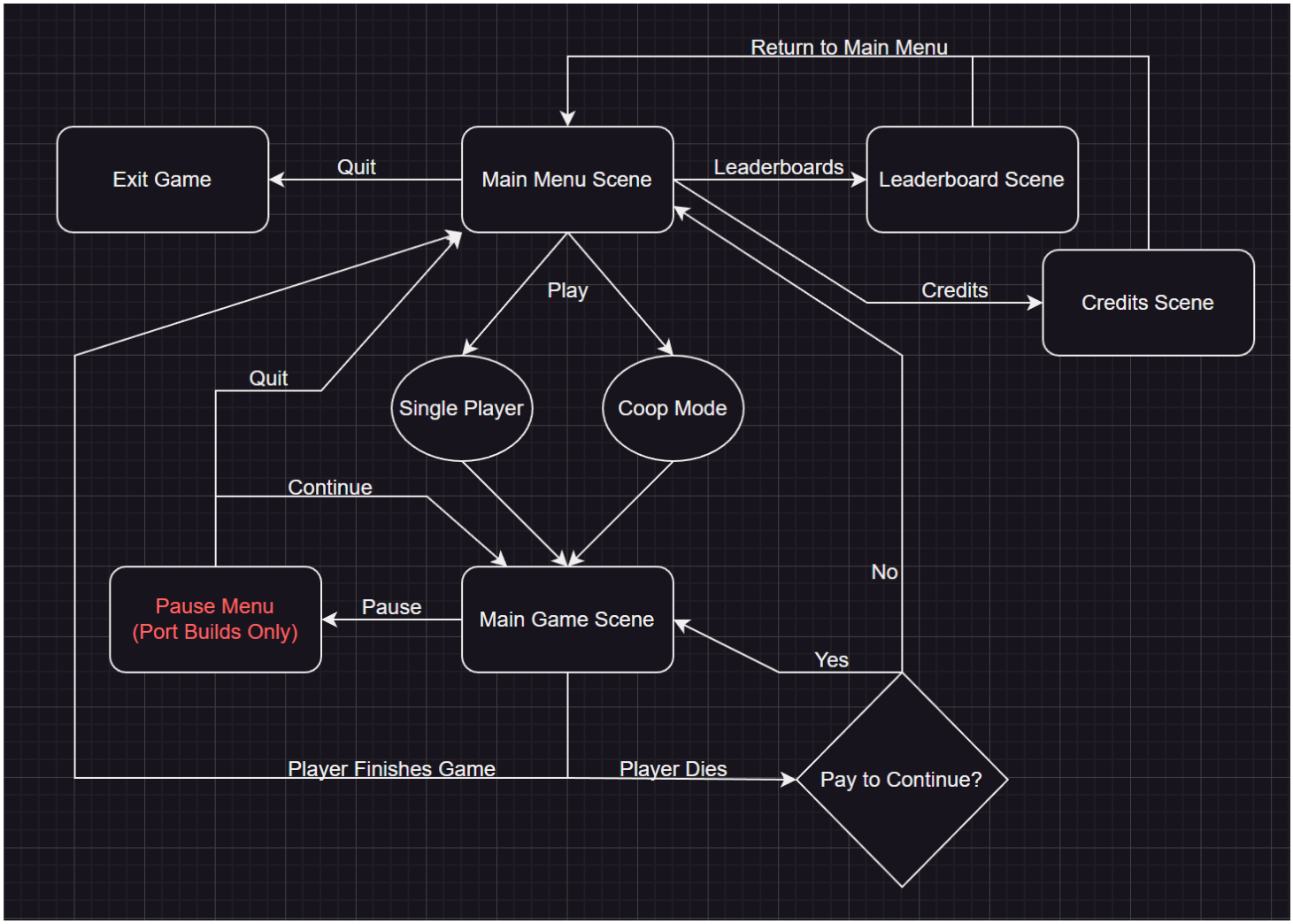
Source Control Repository: *Perforce*

Source Control Client Tools: *Visual Studio*

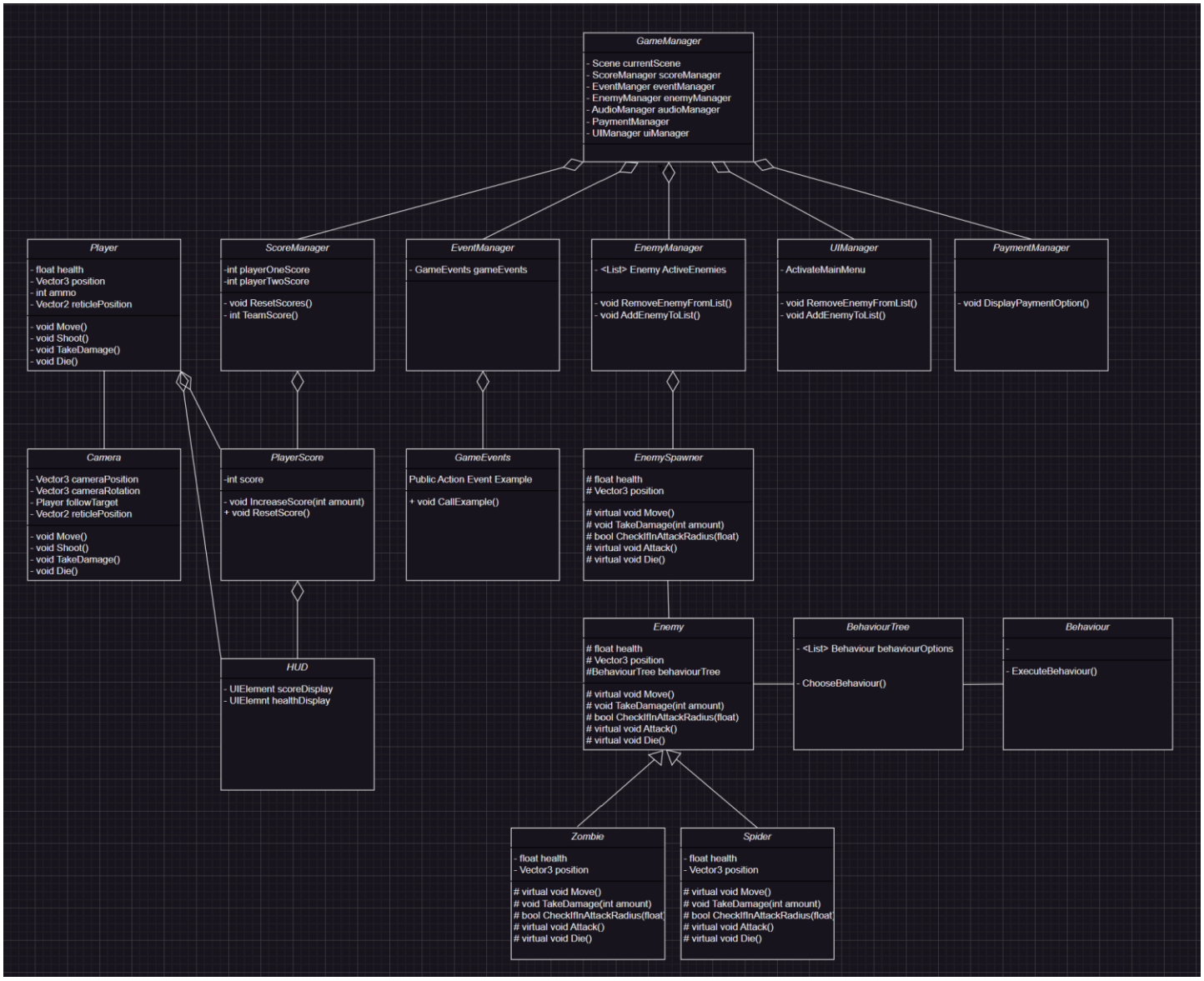
Game Flow

List the scenes in the game, and a short description of what the scene is responsible for.

Scene	Who's responsible	What is does
Enter each game menu		
Main Menu	Brock/Tom	Starting menu, starts a new single or multiplayer game, or exits
Leaderboards Menu	Brock/Tom	Menu for checking high scores, return to main menu
Main Game Scene	Brock/Tom	Main game loop, contains all gameplay, environments, enemies. Handles pay to continue, returns to main menu if game is complete or player does not wish to continue
Credits Scene	Brock/Tom	Rolls credits, return to main menu


Return to Main Menu
Exit Game
Quit
Main Menu Scene
Leaderboards
Leaderboard Scene
Credits
Credits Scene
Play
Quit
Single Player
Coop Mode
Continue
No
Pause Menu
(Port Builds Only)
Pause
Main Game Scene
Yes
Player Finishes Game
Player Dies
Pay to Continue?

Game Objects and Scripts



Further Information:

The diagram provides a structural representation of a game's various components and their interactions. Central to this ecosystem is the GameManager, responsible for overseeing elements such as the current scene, score manager, event manager, and others. The Player entity, armed with attributes like health and position, is equipped with actions like Move(), Shoot(), and TakeDamage(). The Camera interacts closely with the player, offering a perspective defined by its position and rotation. Player's scores are managed by the PlayerScore component, which can increase or reset the score.

In parallel, the game features an EnemyManager that maintains a list of active enemies and facilitates their addition or removal. Individual enemies, such as Zombie and Spider, derive from the core Enemy class, inheriting properties like health and position. Each enemy type can possess unique behaviours or shared actions, as denoted by their virtual functions. These behaviours are further structured using a Behaviour Tree that helps determine which action or behaviour the enemy should execute next.

User interaction is facilitated through the UIManager, which can activate various menus and manage elements like health displays. Furthermore, the EventManager deals with in-game events, which can be triggered or listened to by various other components. Lastly, the PaymentManager is a utility component responsible for handling in-game purchases or transactions.

Having this visual layout of a script structure will aid the team in modularly developing, maintaining, and scaling the game's codebase.

Gameplay Systems

Describe in more detail your individual gameplay systems, including how they interact with other systems in your game, what classes are responsible, and using UML diagrams where appropriate. Eg. If you are developing a random generation system, then describe how it works! If you're developing some unique AI or algorithm for a Goblin Sorcerer, then describe the details of how that will be implemented.

Gameplay System 1 – Player Interactions

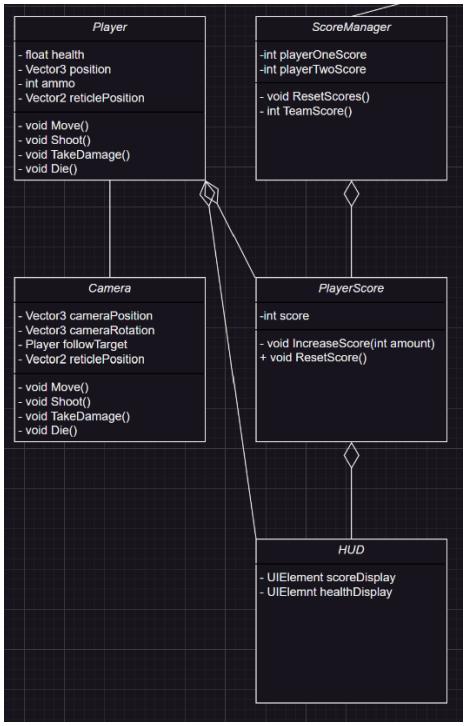
Who's Responsible: *Brock*

Description: *The player represents the main character in the game, with properties like health and in-game position. They can perform various actions like moving, shooting, and taking damage. The camera's main function is to follow the player, providing a perspective based on its position and rotation. It also interacts directly with the player, updating its viewpoint based on player movement and actions.*

Interactions:

- **ScoreManager:** The player's actions, such as eliminating enemies, may contribute to the score which is managed by the ScoreManager.
- **EventManager:** Certain player actions may trigger in-game events that are managed and broadcasted by the EventManager.
- **HUD:** Displays player's vital stats like health.

Diagram:



Gameplay System 2 – Enemy Management

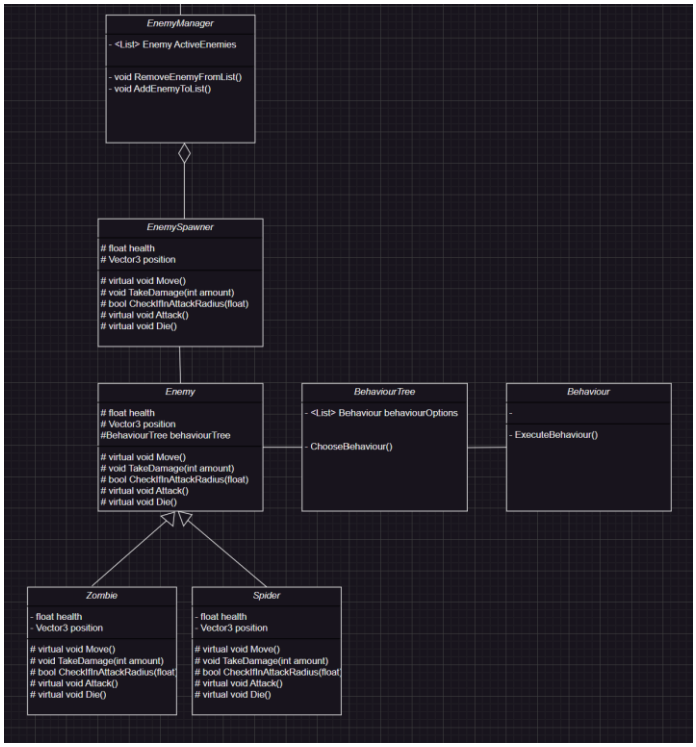
Who's Responsible: *Tom*

Description: *The Enemy Management System handles the spawning, tracking, and behavior of enemies in the game. The diagram depicts a hierarchy of enemies, including generic ones and specific types like Zombies and Spiders. These enemies inherit properties like health and position, but each has unique behaviors and actions. The Behavior Tree plays a crucial role in determining which actions or behaviors the enemy should execute based on the game state or player's actions.*

Interactions:

- **GameController:** Manages the overall game state, influencing how and when enemies are spawned.
- **Player:** Enemies aim to challenge the player, thus their behavior is often in response to player actions.
- **ScoreManager:** The elimination of enemies contributes to the player's score.

Diagram:



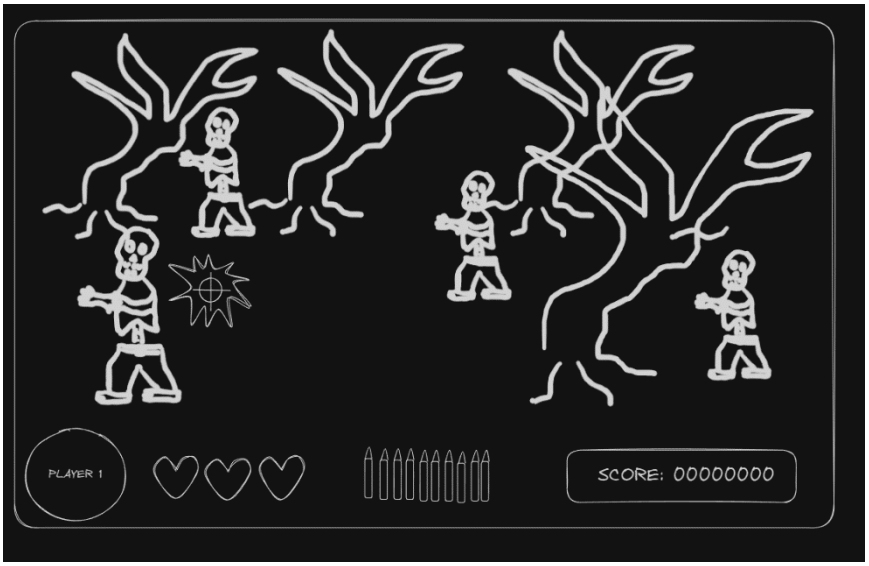
Input Method(s)

Describe the Input method for each target platform (e.g PC / VR / Console).

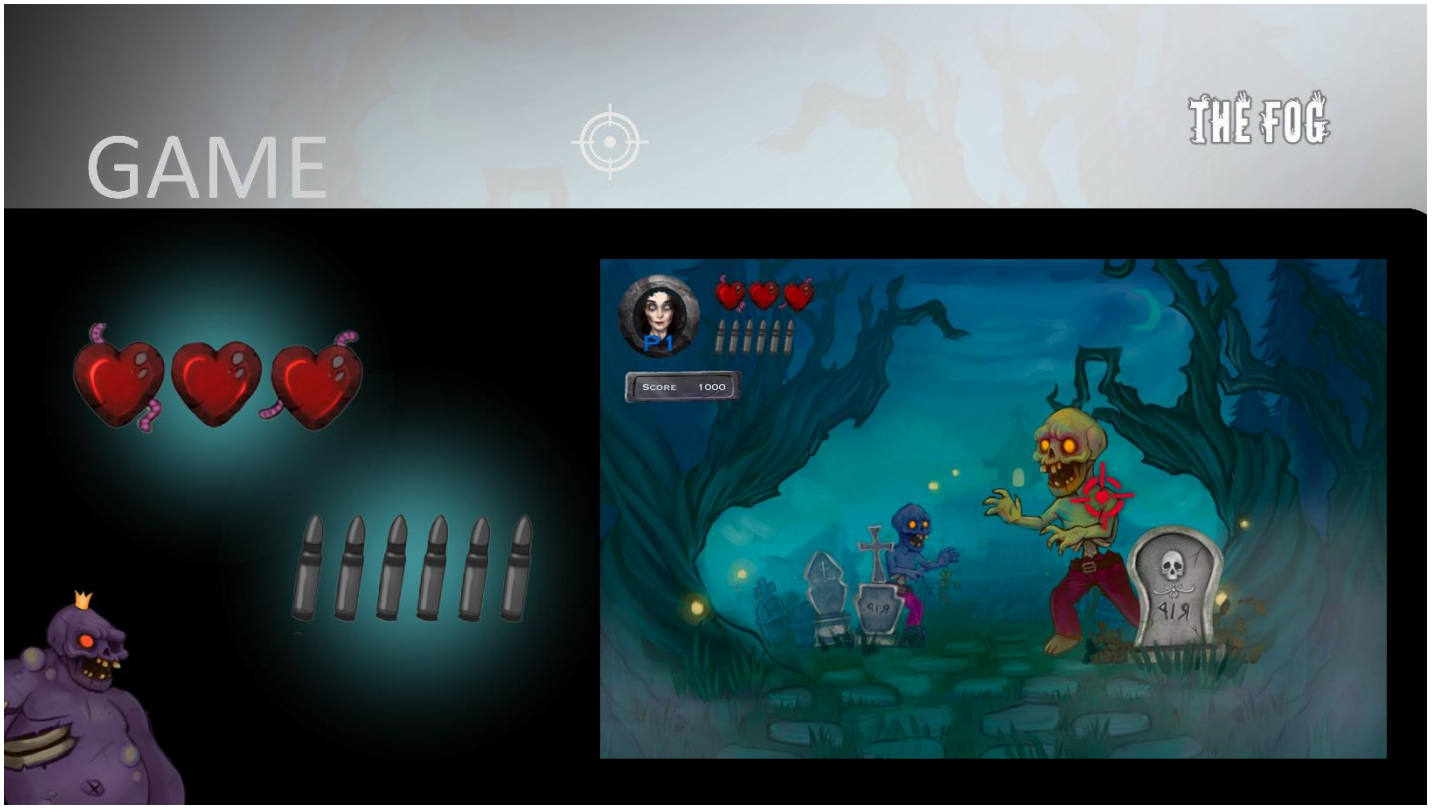
Target Platform	Input System	Who is responsible
Enter each platform/input		
Arcade Cabinet	Arcade Gun with buttons	Brock/Tom
PC	Mouse/Keyboard	Brock/Tom

User Interface

Mock-up/concept of main menu



Sketch of gameplay UI



Concept of gameplay UI and UI elements