# Analysis

Stage 1 & 2 – intro the idea:

The base idea for this project is to create a first-person shooter game that differs your average FPS game by increasing complexity/difficulty. The user will operate a gun and shoot enemies that are attempting to attack the player, however they must use the gun as if it were one in real life – rather than just clicking R to reload they will have to manually do every aspect of reloading a gun such as taking out the magazine, loading rounds etc. I believe there aren’t enough FPS games that significantly differ from each other; though weapons/maps/art-styles may vary, the base concepts of running around and looting areas for weapons to fight enemies in an arena that shrinks in size is used in more than enough of the well-known/common shooter games. This solves the problem of there not being enough first-person shooters that are complex and different to commonly played ones. This would appeal to teenager gamers aged 13-18, who already play shooters and want to try something new that tests their skill, memory and accuracy in a more complex way. To improve my game further I could add different guns that each operate differently, multiple levels/arenas, varying difficulties and a designated tutorial.

I had the idea for this application as I often play many shooter games myself and want a new and unique twist instead of your average game. This idea is not based on solving a traditional problem like repetitiveness or simplicity but instead looks to address a gap in the market: the lack of complex, realistic FPS games that require skilful manual input. The purpose of the project is to offer players challenge by changing the way they interact with weapons, focusing on new skill-based mechanics. In addition, this could also be potentially used as an educational tool, teaching users how firearms operate in a safe digital environment.

My proposed solution is just the idea for my project, and it may change throughout development due to a variety of factors such as stakeholders whose needs are likely to influence my design decisions, my analysis/research, in which I might find ideas that are better than the ones I came up with and use them to improve my game. Limitations such as time constraints are likely to affect the number of working features I can implement in my solution.

Stage 3 - research:

**Reloader: subject\_alpha** is a FPS prototype that focuses on realistic weapon mechanics. The game features an implementation of manual reloading, where players carry out several distinct steps (e.g. removing magazine, insert rounds) to reload. I chose to research this game as it aligns closely with my intended idea of creating a more immersive FPS experience with realistic weapon control.

(https://store.steampowered.com/app/1830160/Reloader\_subject\_alpha/, downloaded 30 june 2025)A screenshot of a video game

Description automatically generated

Ideas to include:

* Manual reloading system – the reloading mechanic is broken down into steps. This concept is essential for my project as it adds realism, forcing players to learn timing and sequence.
* Keyboard-controlled reloading – Keybind based interaction is more engaging than a single button reload.

Ideas to drop:

* UI complexity – the interface is minimal and at times unclear. I plan to implement a more intuitive HUD to display ammo and reload prompts.
* Lack of tutorials – Reloader offers limited instructions to guide users, however I want to include a tutorial of sorts to ease players into the complex mechanics.

**Receiver** is a FPS that focuses almost entirely on simulating firearm mechanics. Each part of the gun (slide, magazine, safety) is independently operated via keyboard keys. The game offers little help – expecting them to learn how the guns work through trial and error.

(https://store.steampowered.com/app/234190/Receiver/, downloaded 30 June 2025)A video game of a video game

Description automatically generated

Ideas to include:

* Realistic interaction – I will mimic the idea of multi-step reloading but scale it back slightly to ensure the game is challenging but still playable for a wider audience.
* High stakes gameplay – The idea that failing to reload properly can cost the player their life creates intense gameplay. This will be used to raise difficulty and reward precision.

Ideas to drop:

* Steep learning curve – The lack of instructions leads to frustration. My game will introduce the mechanics gradually, likely through an early level tutorial.
* Permadeath – I want my game to be challenging, not punishing. Players should be able to retry without fully restarting the game.

**COD Zombies** is a popular FPS that tasks players with surviving against waves of increasingly powerful enemies. It uses simplified gun mechanics but includes strategic features like barriers, weapon upgrades and health indicators.

(https://www.reddit.com/r/CODZombies/comments/qo6t0f/cod\_zombies\_hud\_deevolution\_bo3\_vanguard/, downloaded 30 June 2025)Ideas to include:

* Wave-based progression – I will adopt a similar wave mechanic, where each level gets progressively harder. This gives the players a clear sense of progress and challenge.
* HUD layout – COD uses clean, well-designed UI to show ammo, score, wave number and player health. I plan to implement something similar for clarity and ease of use

Ideas to drop:

* Unrealistic reload system – uses a one button reload, which doesn’t align with my goal. Instead, I will focus on detailed reloading to set my game apart.
* Focus on power-ups and upgrades – My game prioritises realism and challenge through mechanics, so I will avoid features like max ammo pickups or insta-kill powers.

Stage 4 - stakeholders:

My initial stakeholders include a small group of friends and family who enjoying playing video games – particularly FPS games. They will be involved during development to give me feedback on the design choices such as control schemes, difficulty and the realism of the gun mechanics. They will also help test different versions of my project and provide feedback on the usability and challenge level.

I currently have 6 stakeholders aged 14 -18, with a mix of gaming experience. This variety is useful as it allows me to receive feedback from both casual and experienced gamers. I expect their feedback to help shape the UI, control scheme and gameplay features. For example, if most stakeholders find the advanced reloading too difficult, I could add an easier mode with fewer steps.

After development, my broader stakeholder group will be people who are interested in FPS games with more depth/realism, particularly players looking for a new challenge or something different from standard shooters. This group is most likely made up of teenagers and young adults aged 13-25 who are already familiar with keyboard and mouse controls.

To help guide my development process, I will also conduct a survey to find out what they expect from an FPS game, what level of realism and challenge they prefer and which features would improve their experience.

Survey questions:

1. How often do you play FPS (first-person shooter) games?

This will tell me how experienced the stakeholder is with FPS games and help me judge how valid and relevant their feedback is to my project.

1. What do you enjoy most in FPS games?

This will tell me what elements of FPS games are most important to players, and help me decide which features to prioritise

1. How important is realism in FPS games to you?

This will tell me whether players value realism and if my idea for complex reloading is something they would find appealing or frustrating

1. Would you be interested in a game where you manually reload weapons (e.g. remove magazine, insert bullets)?

This will tell me how well my core idea (manual reloading) will be received, and whether I need to simplify or offer different difficulty settings

1. What level of difficulty would you prefer in a game like this?

This will tell me what difficulty I should design the game for and whether to include adjustable difficulty options

1. How important is a tutorial at the start of the game to explain controls?

This will tell me whether I need to invest time into building a detailed tutorial, or if simple prompts and instructions are enough

1. Would you prefer multiple weapons with different reloading styles or just one well-developed weapon?

This will tell me how much variety players expect and help me manage time by focusing on either one gun or multiple guns

1. How should the game display your ammo and reloading status?

This will tell me how to design the UI and HUD so it fits what players are most comfortable with or find most appealing

1. Would you like the game to have wave-based enemy progression (e.g. each round gets harder)?

This will tell me if a progressive difficulty system (like COD Zombies) will appeal to players and or if a level-based system would be more appropriate

Questionnaire results:

Q1: How often do you play FPS (first-person shooter) games?

Responses: Every day (2), A few times a week (3), Occasionally (1), Never (0)  
Analysis: The majority of stakeholders play FPS games regularly, suggesting that their opinions are reliable and come from a place of familiarity with the genre. This means I can confidently use their preferences to guide decisions on difficulty, control schemes and interface features.

Q2: What do you enjoy most in FPS games?

Responses: Fast-paced action (5), Realistic mechanics (2), Customisation (1), Strategy (1)  
Analysis: Most players prioritise fast-paced gameplay, meaning the game must still feel intense even with slower manual reload mechanics. While realism was not a top priority, it still received some votes, justifying its inclusion. I’ll aim for a balance between realism and fluid gameplay.

Q3: How important is realism in FPS games?

Responses: Very (0), Somewhat (5), Not important (1)  
Analysis: None of the players rated realism as essential, but five said it’s somewhat important. This confirms that manual reloading should be present but not overly complex. The realism should add challenge and interest without making the game frustrating or slow.

Q4: Would you be interested in a game with manual reloading (e.g. remove mag, insert bullets)?

Responses: Yes (3), Maybe (3), No (0)  
Analysis: All respondents were open to the core mechanic of manual reloading. However, half of them said “maybe”, suggesting that the difficulty needs to be manageable. This highlights the importance of including a good tutorial and possibly an easier game mode.

Q5: What level of difficulty would you prefer in a game like this?

Responses: Medium (4), Hard (2), Easy (0)  
Analysis: Most stakeholders want a balanced challenge, with a few seeking a more skill-based experience. No one wanted an easy mode, which confirms that the target audience is willing to engage with more demanding gameplay – as long as it is fair and intuitive.

Q6: How important is a tutorial at the start of the game?

Responses: Essential (1), Very (3), Somewhat (2), Not (0)  
Analysis: All respondents see a tutorial as important, with the majority selecting "very". This reinforces the need to include a clear and user-friendly tutorial that introduces reloading and combat mechanics step by step, especially given the game’s complexity.

Q7: Would you prefer multiple weapons or one well-developed weapon?

Responses: Multiple weapons (1), One weapon (1), Doesn’t matter (4)  
Analysis: Most stakeholders don’t have a strong preference. This gives me flexibility during development. I can start with one detailed weapon and potentially later add a second weapon with a different reloading style to provide variation.

Q8: How should the game display ammo and reload status?

Responses: Number/text (5), Visual indicators (3), Audio (1), No HUD (0)  
Analysis: Text-based indicators are clearly preferred, followed by visual UI. I will focus on clear on-screen text to show current ammo and reload status, possibly with a simple icon or colour bar for added clarity. Audio alone is not sufficient.

Q9: Would you like wave-based enemy progression (e.g. each round gets harder)?

Responses: Yes (3), Unsure (3), No (0)  
Analysis: Half the respondents want wave-based gameplay, and the other half are open to it. No one was against it. This supports my plan to implement a wave system that gradually increases difficulty, giving players a sense of progression and challenge.

Stage 5&6 - Requirements:

Requirements and Success Criteria:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Number | Feature | Sub-feature | Explanation | Justification | Importance |
| 1 | Manual reloading | Multi-step input | The player must reload their weapon through 4 steps (eject mag, insert rounds, load mag, cock gun). Each action is mapped to a key. | This is the core mechanic that makes the game unique and more realistic. It increases difficulty and requires memory/skill. | Essential |
| 2 | User Interface (UI) | Ammo, reload, health display | The game will show ammo count and health using text (e.g. Ammo: 3/6, Health: 75/100) | Based on survey results (Q8), players want clear indicators. Essential for usability. | Essential |
| 3 | Enemy Waves | Progressive difficulty | Enemies spawn in waves, with each wave increasing in number and/or speed. At least 3 waves will be created. | Adds a layer of challenge; supported by stakeholder interest (Q9). | Essential |
| 4 | Navigation Menu | Start Game / Tutorial buttons | The player will access the game through a main menu with options to play or view controls/tutorial. | Ensures easy navigation and is essential in any complete game. | Essential |
| 5 | Tutorial / Instructions | Controls & reloading explanation | The player can view instructions about controls and how to reload before playing. | Survey (Q6) shows players need this due to the unusual reload mechanic, reducing confusion. | Essential |
| 6 | Player Movement & Shooting | WASD + Mouse control | Players can move using keyboard and shoot with the mouse. | Core control mechanic for FPS games, no game can function without it. | Essential |
| 7 | Health System | Damage & Game Over | Players take damage when hit by enemies. Health reduces until game over screen appears. | Adds risk and challenge to gameplay. Increasing immersion. | Essential |
| 8 | Weapon variety | Multiple guns | Create at least 2 weapons with different reload steps (e.g. pistol vs shotgun). | Adds variety, stakeholders showed flexibility (Q7). | Desirable |
| 9 | Visual Feedback | Health bars / reload icons | Supplement text UI with visual indicators (bars/icons) for health and reload status. | Supports players who prefer visual over text (Q8). | Desirable |
| 10 | Audio feedback | Empty click / reload sounds | Add sound when a player fires without ammo or reloads. | Adds realism, helps player know when they made an error. | Desirable |
| 11 | Pause Functionality | Resume / Restart options | Pressing ESC opens a menu allowing the game to pause and give resume/restart buttons. | Improves playability, helps testing and debugging. | Desirable |
| 12 | Progress Feedback | Wave summary screen | At game over, display wave reached (e.g. “You reached wave 5”). | Adds satisfaction and motivation to replay, simple to implement. | Desirable |
| 13 | Settings Menu | Toggle sound / adjust keys | Include options to mute sound or remap keys. | Allows personalisation and improves accessibility. | Desirable |
| 14 | Enemy AI | Basic pathfinding | Enemies follow player’s position using simple algorithms. | Adds challenge and increases engagement. | Desirable |
| 15 | Hardware | Desktop PC with keyboard and mouse | The game will be developed and played on a desktop using a standard keyboard and mouse input. | Unity is designed for desktop development; most FPS games require a keyboard and mouse. School PCs meet the minimum requirement. | Essential |
| 16 | Software | Unity Game Engine | Unity will be used to design, develop, and test the FPS game using C#. | Unity supports 3D development, physics, UI tools and cross platform builds – ideal for making an FPS. | Essential |
| 17 | System Requirements | Minimum 8GB RAM, basic dedicated GPU | Unity projects run best on systems with moderate specs. | Ensures that development and testing can occur smoothly without lag or crashes. | Essential |

Limitations:

There are several potential features that I have chosen to leave out, below are the key features I considered but will exclude from my final solution:

1. **Online multiplayer**

**Explanation**: An online multiplayer mode where 2+ players can play at the same time together.

**Justification**: Implementing networking in Unity will require many features such a server, data sync and possibly external services. This also significantly increases the complexity of testing, programming and debugging. As this is a solo project with limited time and hardware access, this feature is outside the project’s scope.

1. **Complex AI**

**Explanation:** Intelligent AI enemies that can conduct complex behaviour.

**Justification:** I intend to implement simple enemy behaviour (e.g. basic chasing or moving towards the player) but an advanced AI enemy would require a more detailed and complex map and a large amount of coding. My focus is on manual reloading, so AI complexity is not a top priority.

1. **Weapon customisation**

**Explanation:** A system that allows weapon customisation with scopes, suppressors, magazines etc.

**Justification:** While this would add replay value, this system adds levels of interface design and weapon model variations. Since the project already includes multiple reload steps and possibly multiple weapons, adding customisation could lead to over scoping.

1. **Inventory**

**Explanation:** A system where players can select and change between weapons/items.

**Justification:** This requires additional unnecessary UI whereas the manual reloading system already has significant complexity, so including an inventory would compromise development time.

1. **Cinematic elements**

**Explanation:** A story or cinematics like cutscenes.

**Justification:** While a story could increase engagement, it is not the focus of the experience I am aiming to deliver. As my focus is on gameplay mechanics, adding story elements would increase workload without directly supporting the main goal.

1. **Cross-Platform compatibility**

**Explanation:** Allowing the game to be played on different platforms

**Justification:** While Unity can export to mobile, adapting controls for touch screens and testing across devices could lead to many complications. Cross platform support would be unnecessary and would take focus away from the main experience.

Stage 7 – Computational Methods:

My solution is highly suitable to be solved using computational methods as it involves simulating real world processes and handling interactive systems. Computers excel at managing processes that require precise control, user interaction, and graphical output – all of which are essential in my project. The solution uses multiple computational methods to execute the complex features involved in the game.

**Abstraction** - This project uses abstraction to simplify real weapon features and processes into a playable system. In real life, reloading a weapon involves several actions, many being too detailed or complex to replicate directly in code. Instead, the system focuses only on the criticalsteps – ejecting the magazine, loading rounds into a new empty one and loading the new mag – and represents them as individual key presses. This allows the game to feel realistic without overwhelming the player with unnecessary details. By removing minor or non-essential elements (e.g. hand animations, bullet types) abstraction ensures that only the relevant/necessary parts of the problem are included in the solution. This makes the solution both implementable and engaging whilst maintaining the core concept of realism.

**Decomposition** – The game is broken into multiple components, each responsible for a key part of gameplay including:

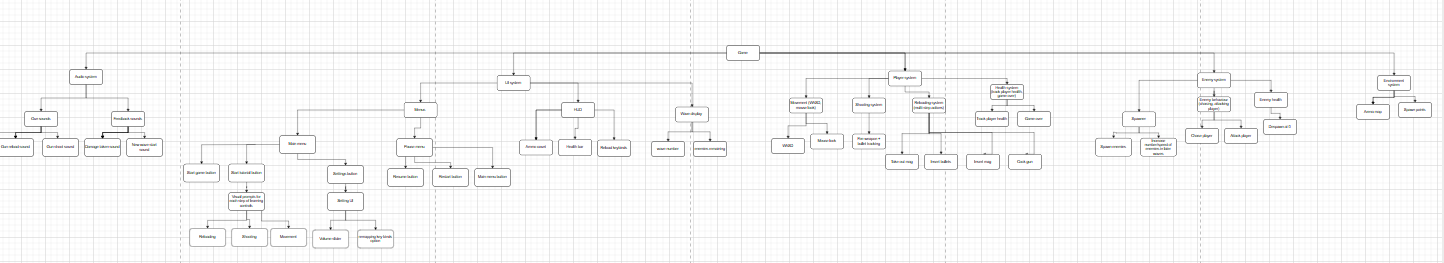
* A player control system (handling movement and aiming)
* A shooting system (firing and bullet tracking)
* A manual reloading system (realistic reload sequences)
* An enemy wave spawner (increasing difficulty over time)
* A health and damage system
* A UI manager (displaying ammo, health and wave number)

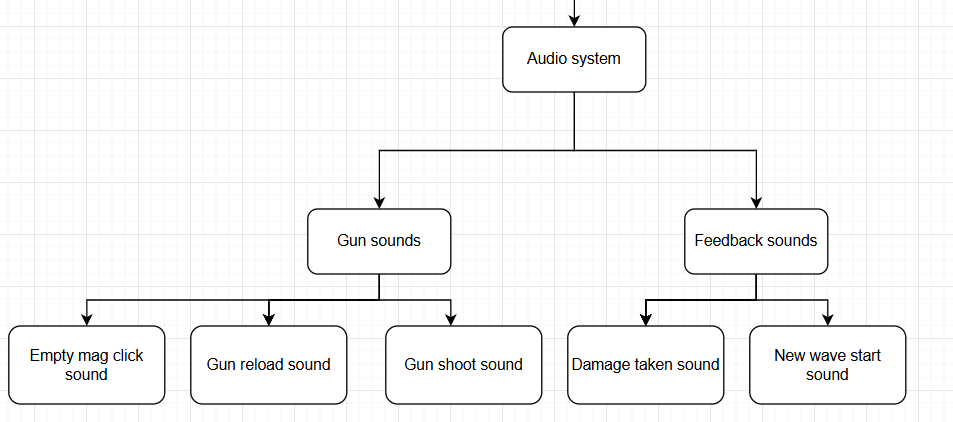
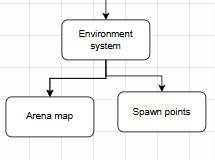
Using decomposition allows each of these systems to be developed individually using separate scripts and classes in Unity. This approach improves maintainability, makes debugging easier, and allows for parallel development of features.

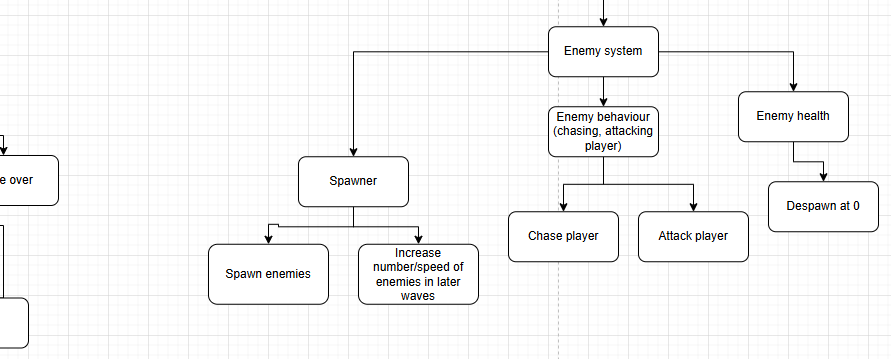
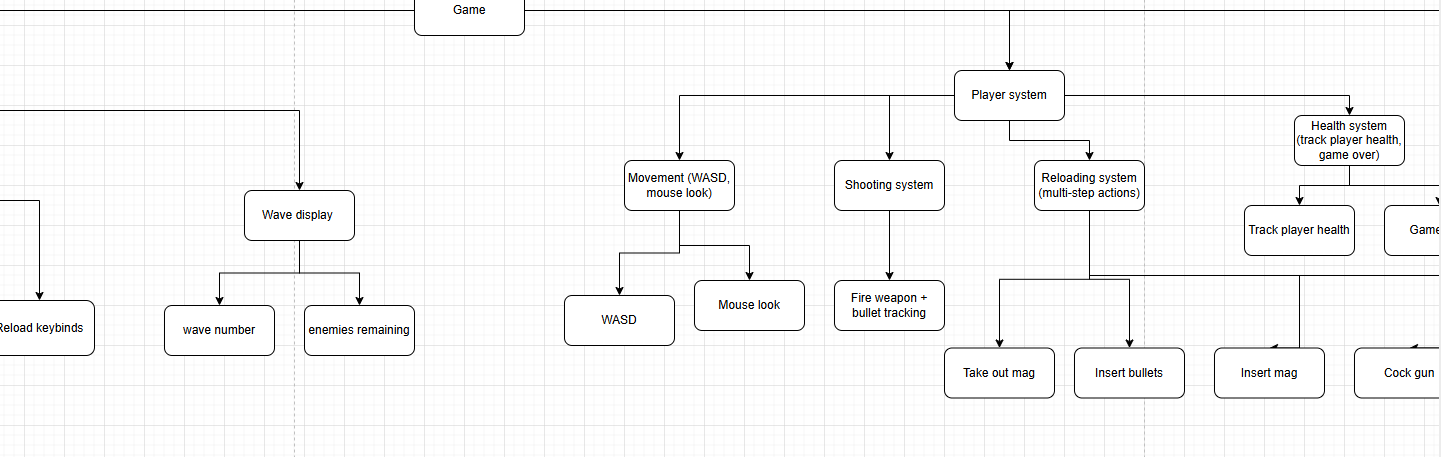
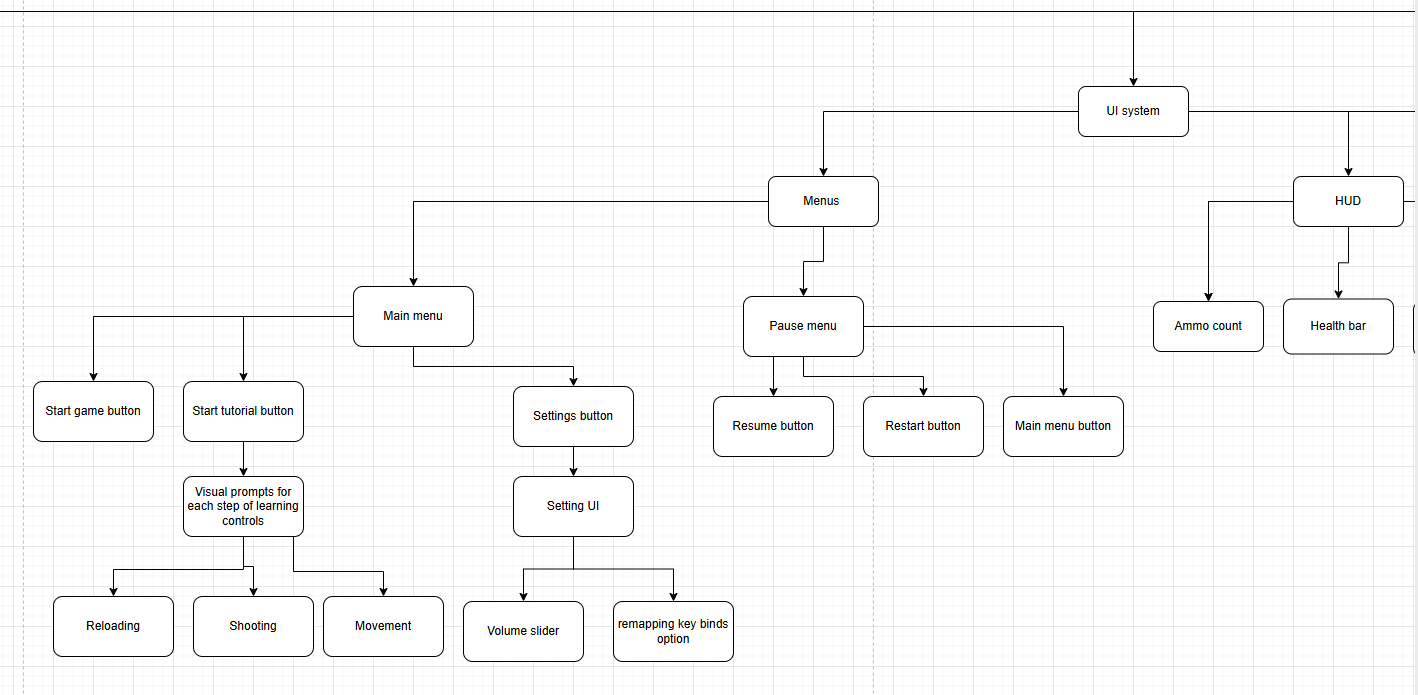
**Concurrency** – The game requires multiple systems to run simultaneously. For example, while the player is reloading, enemies may be moving or attacking, UI elements are updating constantly and the audio system may be playing sound effects. Unity’s game engine handles this through its system, allowing concurrency. Without this, the game would have to process actions one at a time – resulting in lag and input delays. Computers are ideal for handling simultaneous processes, allowing gameplay to feel smooth even when many events are happening at once.

# Design

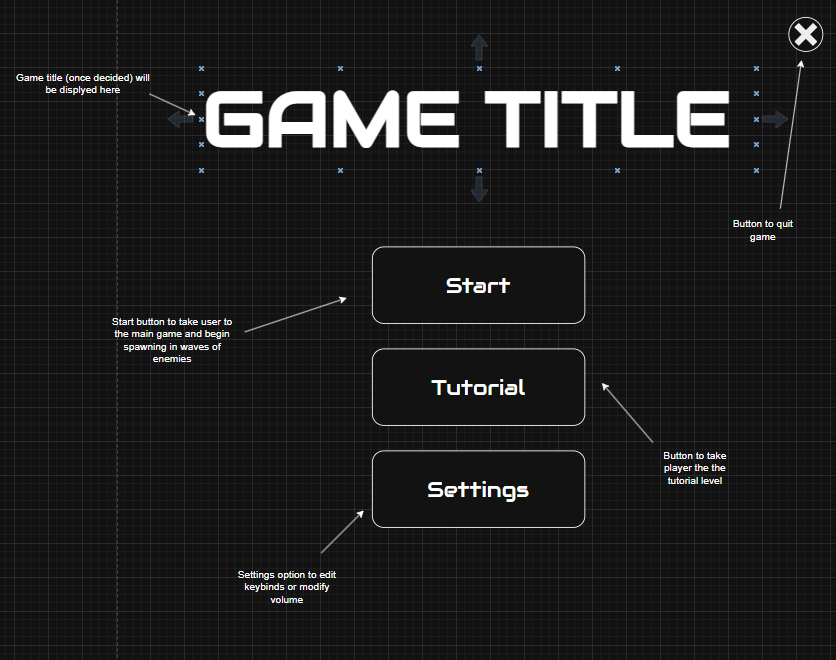
Structure Diagram:

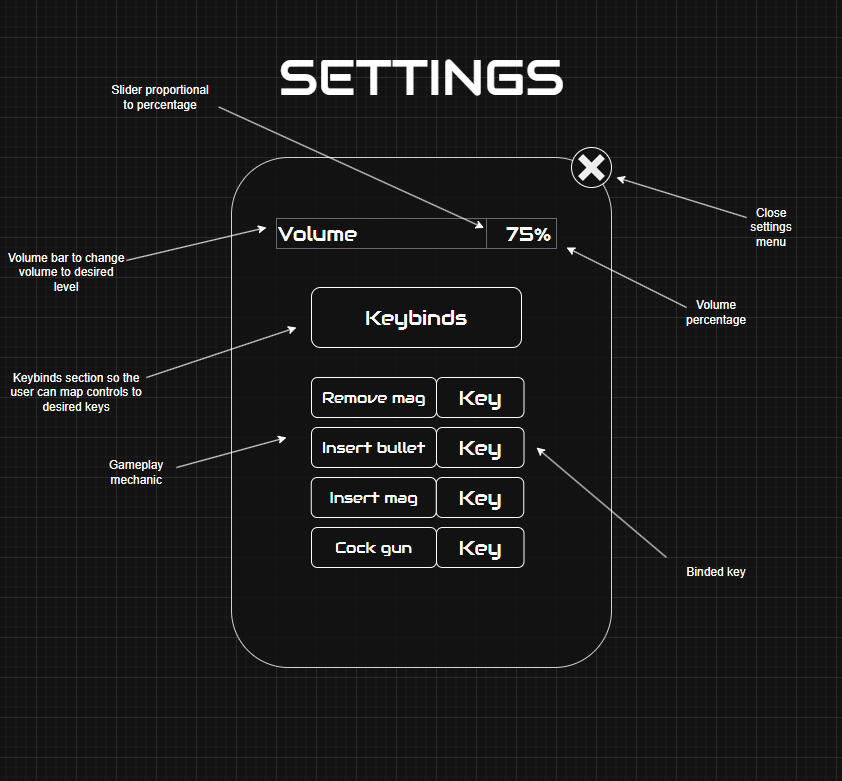
This is the overall structure diagram view – Each part is a separate game system, consisting of: Audio system, Player system, Enemy system, UI system, Environment system.

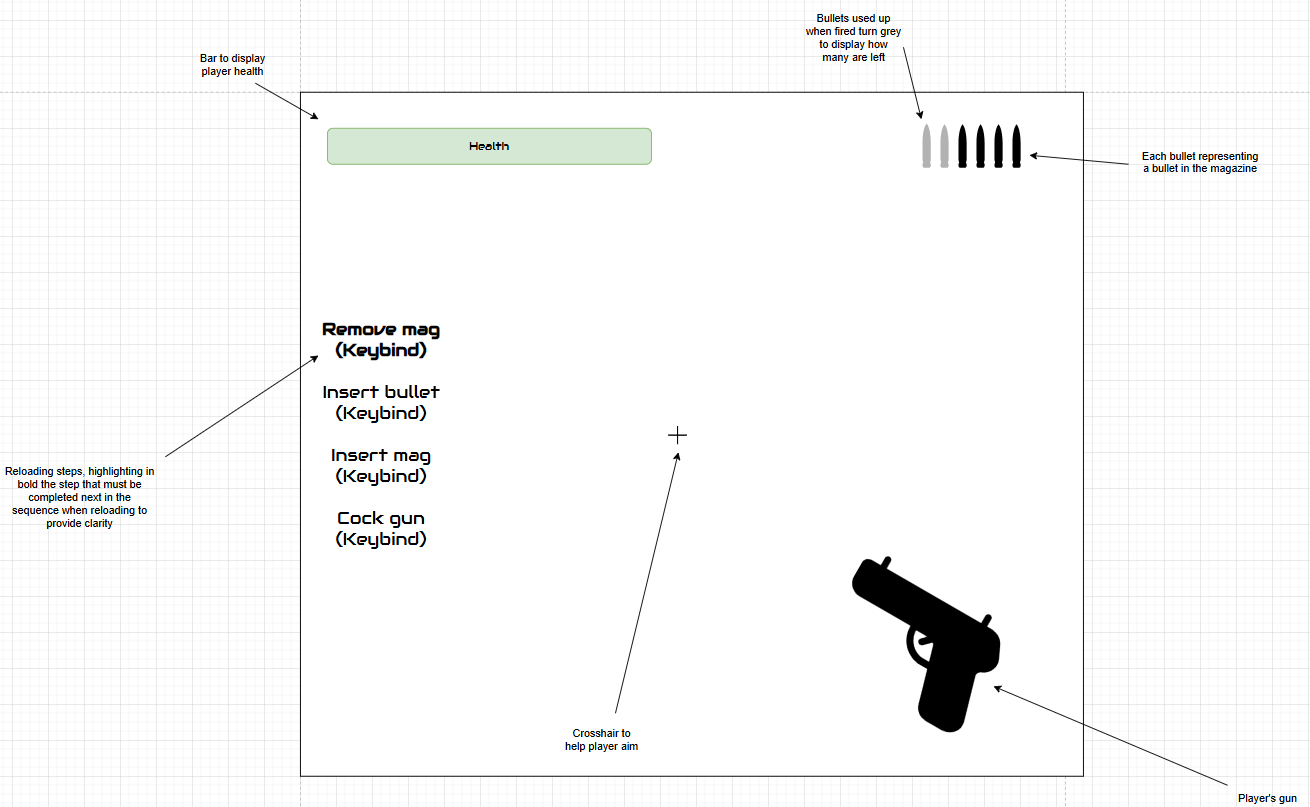
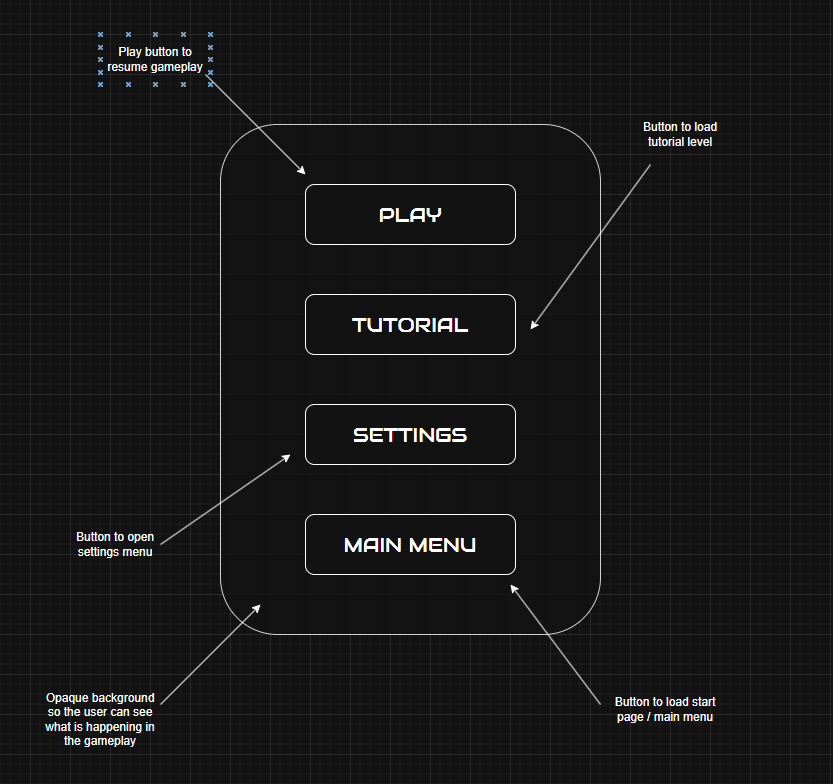


GUIs:

GUI Design: I want the GUI to be straightforward and user-friendly, enabling users – particularly my target audience who are experienced with first person shooters – to easily understand the purpose of each function. As a result, every button is distinctly labelled to indicate its function, and there are very few symbols to represent buttons (only an X for exit – which is universally known). Main Menu – When the user opens the game, this is the first screen that will appear. I have chosen a simplistic design for the menu, with each button in a large rounded rectangle. Once I have decided the game name, it will be displayed in the GAME TITLE area with the same given font. I have chosen this font (Audiowide, Google Fonts) for the menu (and plan to keep it consistent throughout the game) as it is clearly readable, but more stylish than a standard font such as Roboto. There is no current background, as I plan to have a blurred gameplay screenshot as the background for the main menu which can only be implemented later. The buttons on display have been done so the user can easily navigate to any part of the game they wish - without having to go through multiple pages and keeping simplicity. I may make it so the buttons change colour when hovered over with the mouse, however I will ask for stakeholder feedback to see if it would be necessary or the current design is easy enough to navigate.



Settings menu – This will contain options to adjust volume and remap key controls, keeping the same theme of Audiowide font and rounded rectangle (except for the volume slider as it didn’t suit the aim for a sleek, clean design when originally made). Each of the modifiable keys are clearly shown in large boxes, which will be binded to the default controls when the game is launched (F, G, H, J)

Gameplay UI - I have chosen to have simple icons to represent different features such as bullet icons to display ammo in the magazine (coloured ones representing the available bullets, the lower opacity being used bullets) and a bar to display health. I intend to have a green bar for the health and when taking damage, the bar will slowly turn red (from right to left) to give clear, visual displays for how much health the user has. Visual indicators were desired from stakeholders, so this is why I have chosen to use icons instead of simple text with a number to represent bullets and health. However text was also desired, so I chose to have the keybinds for reloading on the side, with the current step in the reload process highlighted in bold to provide clarity for the user. The gun (like most common FPS games) is in the bottom right corner, and will eventually be animated to provide a secondary visual indicator of what action the player is taking (e.g. shooting, removing the mag), with the crosshair being a small + in the centre of the screen to help aim accurately. I will ask for stakeholder feedback on whether they would like a different crosshair or other UI features once basic gameplay is implemented, to clarify whether the GUI is too crowded/not informative enough.Pause menu – This will appear by pressing the ESC key during gameplay (like any other FPS) and exited via the same key. I have chosen to do this so the game doesn’t differ too much from other FPS games, as the reloading mechanic itself can be quite confusing so I want most features to be self-explanatory and similar to games my stakeholders will be familiar with. Along with this, the menu fits the same style of rounded rectangles with the Audiowide font. I have also chosen the background to be opaque, so the user can see where they are in the game to avoid unexpected surprises as that can cause them to lose the game (especially further in when waves are more fast-paced).

Development plan:

**Core Player Controls**

**Functionality**: Add basic movement (WASD + mouse look) and shooting (click to fire and ammo decreases) so the player can move around and fire

**Justification**: This is the foundation of the game and nothing can be tested if the player cant move or shoot, so it must be the first stage

**Deadline**: Week 1-2

**Initial test plan**:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test number | Valid/invalid | Description | Test data | Expected result |
| 1a | Valid | Player moves when given input | W, A, S, D keys pressed and held | Player makes continuous movement forwards (W), left (A), backwards (S), and right (D) |
| 1b | Valid | Spacebar will cause the player to jump | Click spacebar | Player jumps up then falls down |
| 1c | Invalid | Player will not be able to jump unless they are on the ground | Jump while in the air | Player will not jump again and fall as intended |

**Reloading System**

**Functionality**: Create the multi-step reload (take out mag, insert bullets, insert mag, cock gun), each mapped to a key.

**Justification**: This is the main unique feature of the game so it is essential to get working early to make sure it’s testable

**Deadline**: Week 3-4

**Initial test plan**:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test number | Valid/invalid | Description | Test data | Expected result |
| 2a | Valid | Reload with given keybinds in order will cause ammo to reload | Keyboard click F, G, H, J | Ammo will go from 0 to 6 |
| 2b | Invalid | Reloading in the wrong order will not increase ammo | Keyboard click combinations of F, G, H, J in different orders (except the original) | Ammo will remain at 0 |
| 2c | Invalid | Attempting to shoot while reloading will be disabled | Left mouse click at all stages during the reload process | Gun will not shoot |

**Map/Arena**

**Functionality**: Create a simple 3D arena for the player to walk around in

**Justification**: Player needs space to walk around to make testing realistic. The map must exist before adding enemies as they need somewhere to spawn and chase the player

**Deadline**: Week 4–5

**Initial test plan**:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test number | Valid/invalid | Description | Test data | Expected result |
| 3a | Valid | Player will be unable to walk through walls | Attempt to walk in all directions through a wall | Wall will block player from moving through it |

**Essential UI**

**Functionality:** Add HUD with ammo, reload status, health and wave counter. Basic menus like main menu and pause menu

**Justification:** The UI makes the game easier to understand and is needed for testing, like checking ammo count, health etc work properly. Menus make navigation easier

**Deadline:** Week 5–6

**Initial test plan**:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test number | Valid/invalid | Description | Test data | Expected result |
| 4a | Valid | Firing and reloading will update HUD displays | Left click, press F, G, H, J in order | Ammo and reload displays will update accordingly |
| 4b | Valid | Taking damage will cause the health bar do go down | Receive damage (from a temporary set of code as enemies aren’t in the game yet) | Health bar will reduce in proportion to damage taken |
| 4c | Valid | Multiple menus will be implemented for ease of navigation | Click on menu buttons to verify the load the correct scene | Corresponding scene will be loaded |
| 4d | Valid | Pausing the game will freeze the game and disable shooting and moving | Keyboard click ESC, then attempt to move and shoot | Gameplay will be disabled while in the pause menu |

**Enemy System**

**Functionality:** Make basic enemies that chase the player and deal damage, add wave spawner that increases number/speed of enemies each wave.

**Justification:** This adds the challenge. Doing it after the player and UI makes sense because then combat can be tested with ammo and reloading mechanics

**Deadline:** Week 6–7

**Initial test plan**:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test number | Valid/invalid | Description | Test data | Expected result |
| 5a | Valid | Enemies will move towards the player and attempt to attack | Spawn an enemy in the game scene | Enemy will move towards player and attack |
| 5b | Valid | Enemies will increase in number and speed as the wave number increases | Play game until waves increase | Enemies will increase in number and speed, increasing as the wave reaches a higher stage |

**Game Over**

**Functionality:** Add game over screen when health hits 0

**Justification:** This finishes the main gameplay loop so it feels like a proper game (to survive as long as possible), added after enemies so it can be linked to their attacks

**Deadline:** Week 8

**Initial test plan**:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test number | Valid/invalid | Description | Test data | Expected result |
| 6a | Valid | At 0 health, the game will be over and load the “Game Over” scene | Wait for enemies to damage player until health reaches 0 | Game Over scene will trigger |
| 6b | Valid | Restart button on Game Over scene will reset game and load player into a new endless mode | Click on “Restart” button | Player will load into a new endless game |

**Tutorial**

**Functionality:** Make a tutorial level that explains the controls, shooting and how to reload in steps

**Justification:** Survey feedback from stakeholders said players need guidance for the unusual reloading mechanics, done later so it reflects final controls and UI

**Deadline:** Week 9

**Initial test plan**:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test number | Valid/invalid | Description | Test data | Expected result |
| 7a | Valid | Tutorial scene will guide player through gameplay mechanics | Click on tutorial button in main menu | Tutorial scene will load and walk through game mechanics |
| 7b | Valid | Each step in tutorial will correspond to where the player is in the game | Play through tutorial | Each mechanic will be highlighted with keybinds until player presses them |

**Polishing & Extras**

**Functionality:** Add extra features like sound effects, wave summary/details, settings.

**Justification:** These aren’t essential for the game to work but make it feel more complete hence leaving them until the end. This means if time runs out the game is still functional

**Deadline:** Week 10–11

**Initial test plan**:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test number | Valid/invalid | Description | Test data | Expected result |
| 8a | Valid | Sounds will play when shooting, reloading, empty gun and taking damage | Shoot, reload, shoot with an empty mag, take damage | Corresponding sound effects will play |
| 8b | Valid | At game over, a wave summary will appear | Reach game over | Wave summary with statistics on enemies killed and damage taken appear |
| 8c | Valid | Options menu will allow volume control and keybind remapping | Navigate through options menu features and modify sound and controls | Corresponding feature will change to set options |

**Testing & Evaluation**

**Functionality:** Final tests of everything against success criteria, fix bugs and get final feedback from stakeholders

**Justification:** The final stage to ensure the game is complete, meets success criteria and is functional

**Deadline:** Week 12

**Initial test plan**: Test game against all success criteria

# Development

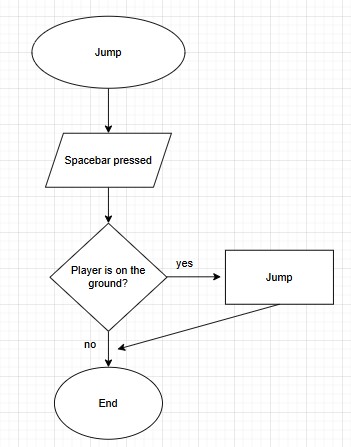
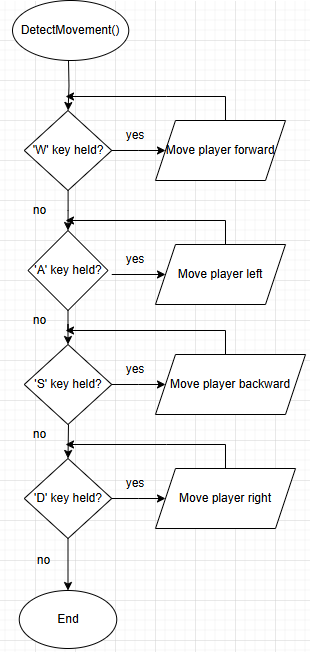
Stage 1 – Core controls:

Minimum success criteria

* Create player character
* Player can move forwards, backwards, left, right
* Player can jump

Stage 1 involves creating the player and giving them the ability to move. This involves moving the player in 4 directions with the corresponding key from W, A, S and D (representing forwards, left, backwards and right respectively). I also intend to add a jump feature where the player can jump if they are on the ground. This stage is key to the minimum viable product.

Implementation:

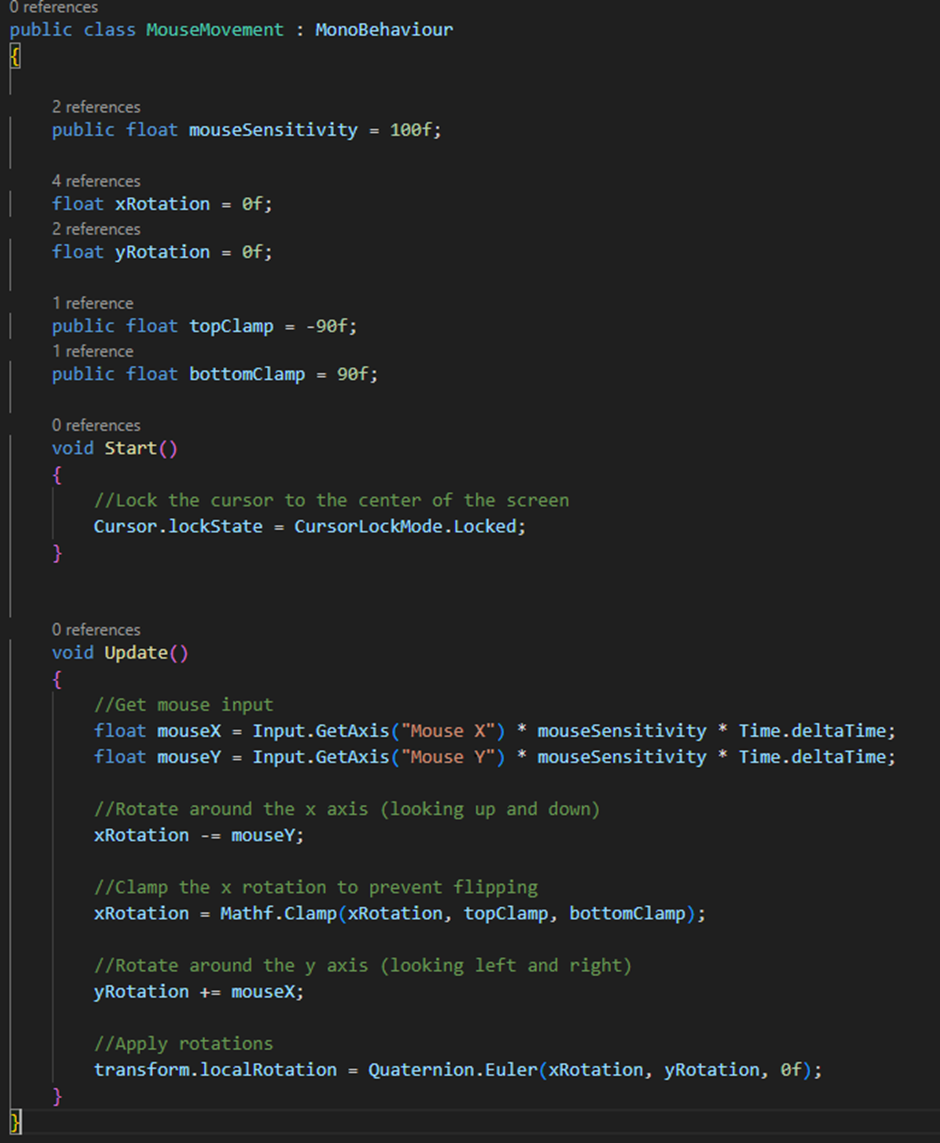


The flowchart for DetectMovement() shows the basic input detection for the keys WASD, which will allow the player to move into their desired direction. I have chosen not to include speed in the diagram as this will be undecided untill I test the program and observe which speed would be most appropriate for the game. The Jump() will be used to detect if the player is on the ground, to prevent unlimited jumps while in the air. These 2 functions will help me create the base of the game, as further development without it would be significantly difficult.

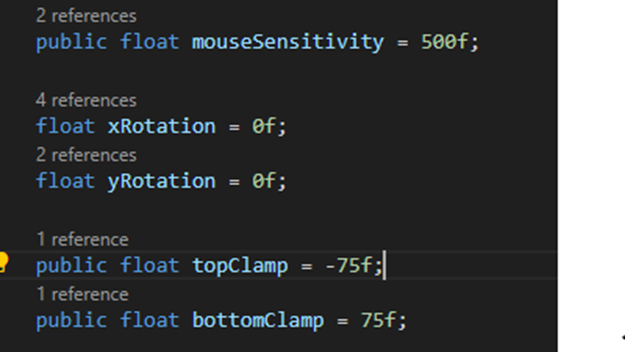
Data dictionary:

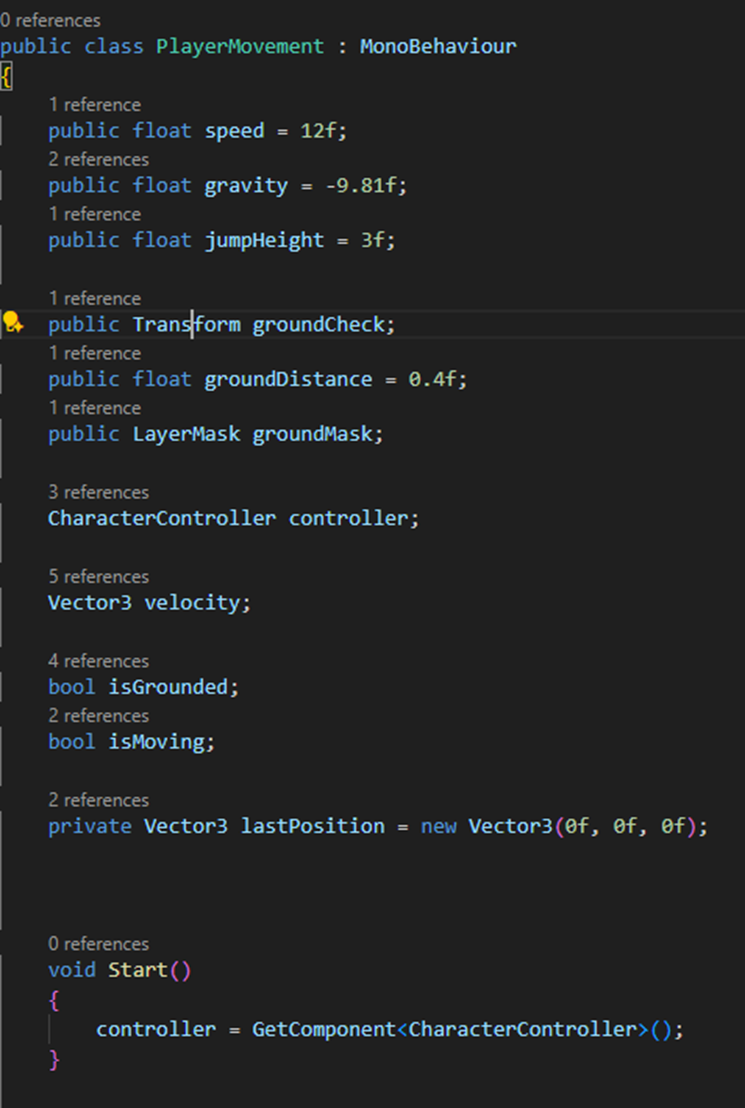
|  |  |  |  |
| --- | --- | --- | --- |
| Variable name | Data type | Description | Validation |
| Speed | Float | Controls how many units of distance the player moves each second | Change the value and observe how fast the player moves |
| JumpHeight | Float | The velocity the player is shot into the air with when jumping | Press spacebar, observe a suitable velocity for a realistic jump |
| Gravity | Float | The velocity that will be put on the player downwards as an act of gravity | When jumping, observe a suitable velocity that the player falls at |

Developing:

MouseMovement handles the first-person camera for the player. It rotates the camera based on mouse input and clamps the vertical rotation to prevent the camera from flipping, locking the cursor to the centre at the start. I have chosen to do this first as it is a core gameplay element and the program won't be testable without it.

I made the camera look separate from the movement so its own script can be changed or reused without touching the players movement to make debugging easier. I added clamps to prevent the camera flipping over itself and looking too far up/down, making gameplay smoother.

I noticed the mouse sensitivity was too low, so I changed the original value from 100 to 500. In addition, when testing the clamps I noticed they were too high/low, so I also reduced these values to stop the player looking directly above and below them for more realistic gameplay.



 PlayerMovement implements the movement using a CharacterController. It reads movements and creates a movement vector, applies gravity and jumping, and using a ground check to determine whether the player is on the floor or not to disable jumping unless floored. I have chosen to do this straight after as it is another core requirement and is essential for most testing.

I used a CharacterController for the player as it automatically handles collisions and step movement which keeps the controls smooth and avoids physics issues. The movement is based on the camera, using transforms so the player moves in the direction they are looking which is essential for a FPS. I also used a sphere check at the players feet to detect when they are on the ground to prevent double jumping, resetting velocity.y to a small negative once grounded to avoid floating/jittering. I also included an isMoving check for features like footstep sounds later in development, and made the important variables such as speed, jump height and gravity public so they can be easily tweaked during testing.

Testing:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test number | Valid/invalid | Description | Test data | Expected result | Evidence |
| 1a | Valid | Player moves when given input | W, A, S, D keys pressed and held | Player makes continuous movement forwards (W), left (A), backwards (S), and right (D) | Core controls -video 1 |
| 1b | Valid | Spacebar will cause the player to jump | Click spacebar | Player jumps up then falls down | Core controls -video 1 |
| 1c | Invalid | Player will not be able to jump unless they are on the ground | Press spacebar while in the air | Player will not jump again and will fall as intended | Core controls -video 1 |
| 1d | Valid | Clamps prevent camera from over rotating | Move mouse in all directions including far up and down | Horizontal rotation will be unrestricted however vertical will be prevented at too high/low | Core controls -video 1 |

Evaluation:

This stage went very smoothly, the issues where variables were misvalued (e.g. mouse sensitivity, bottom clamp being negative and top being positive) were all fixed with relative ease. Regarding success criteria, this satisfies all requirements for this stage as the player can move in all directions, jump, and camera movement clamps were successful; meeting these points in the success criteria is significant.

Stage 2 – Reloading System: