CS/CE Senior Design Specification report

Nuclear Winter

Mina Anis

&

William Lin

Advisor: Dr. Edward Currie

Spring 2018

Table of Contents

Appendix……………………………………………………….2

Chapter 1: Project Scope..............................................................3

Project Goals..............................................................4

Chapter 2: Requirements Specification

System Perspective.................................................5-6

User Stories...........................................................6-11

External Interfaces...................................................12

Non-Functional Requirements............................13-14

Chapter 3: Constraints

Engineering Standards............................................15

Realistic Constraints...............................................15

Chapter 4: Project Design

System Components………………………………16

Structure and relationships………………………..17

User Interfaces…………………………………18-20

Detailed component description……………….21-22

Methodologies…………………………………….23

Design alternatives………………………………..23

Reuse and relationships with other products………23

Resource list……………………………………….23

Resource skill list………………………………….23

Chapter 5: Project Plan

Sprints………………………………………..24-25

Risk………………………………………………25

Estimated financial budget……………………….25

Teamwork Plan……………………………………25

Chapter 6: Project Methods

Technical Methods……………………………..26-28

Tools……………………………………………28

Programming Languages……………………….28

Use of Open Source Code………………………29

Chapter 7: Project Implementation and Results

Implementation Strategies………………………29

Sprints………………………………………… 29-33

Results………………………………………….. 33-36

Chapter 8: Problems and Changes

Problems and Changes…………………………37

Appendix………………............................................................37-39

**Abstract**

Most games in the Battle Royale genre nowadays focus purely on the PvP aspect of shooter games, but we wanted to create a Battle Royale where anyone can have fun and play against bots in a PvE or PvP environment. Our design approach was to use Unity 3D and build a game that would add more and more features as time goes on. With this design methodology, we used sprints as a way of tracking our progress. The biggest problem we encountered during our project was from the unity tool itself. Unity would quite often crash and sometimes cause corrupt data which would lead to loss of work efficiency and also stress. Our game will allow users to set custom games with varying levels of bots and also incorporate the option to choose how many bots they wanted to play against, we also wanted to include cross platforming so that for example, someone on PC and someone on iPhone could play together, but due to time constraints we were forced to scrap that part of the project. Our current goal is to have multiple people be able to play together at the same time with or without bots, while also having a smooth and fun experience in our Battle Royale game. Eventually, we hope to port the game onto mobile devices and possibly allow cross play between PC and smartphones, but that won’t be happening in this semester.

**Chapter 1: Project Scope**

Section 1.1: Project Scope and Vision

The problem we are trying to solve is the lack of variety in games these days. While there are a plethora of third person shooter games out now, almost all of them are online only games, and their offline capabilities are limited. The gaming market is growing exponentially, with third person shooters becoming one of the most popular, if not the most popular genre of games to play today. We feel that by releasing a Battle Royale game with specific offline capabilities that allow the user to enjoy the game even while offline could be our ticket into the market.

The Battle Royale genre is primarily dominated by games such as “Fortnite”, “PUBG”, and “DayZ” at the moment, but these games don’t exactly qualify as “existing solutions” as they provide very little offline capabilities. We feel that taking our own spin on this game by including some bots to have a game mode for the user to do in the case that they have no access to the internet, which is a very common situation.

Our prototypes main feature will be that it includes a basic form of AI that will play “with” you as you play the game. The AI at first will be a set script for the bot to follow, with added delays to facilitate the “difficulty” system that the player may adjust. This is good because normally, as one player you usually would not be able to play a Battle Royale game as you’d need opponents to play against but in our game this problem wouldn’t exist as the AI would fill that role. They would spawn throughout the map should you so choose to play with them and they will function as real players. The “AI” will be basic at first but given proper time we will attempt to transition them into self-learning AI that would gain their skills from watching people play the game.

Section 1.2: Project Goals

1. Nuclear Winter will have both a Player vs Player mode where they’ll be able to play with both other players and bots (if they would like), and a Player vs Environment mode where the player can play against the bots on with all the AI out to get them.
2. Nuclear Winter’s primary development will be on iOS.
3. Given proper time, Nuclear Winter will also have a PC port.
4. Nuclear Winter will have an AI with adjustable difficulty to accommodate players of all skill levels.

**Chapter 2: Requirements Specification**

Section 2.1: System Perspective

**Database**

* Where the userInformation class is stored which contains all unique information about a user’s respective character.
* Will also store the AI behavior patterns.

**Database**

* Where the userInformation class is stored which contains all unique information about a user’s respective character.
* Will also store the AI behavior patterns.

**Game Client**

* Main thing the user will be seeing.
* All aspects relating to the game will be displayed through here.

Figure 1: System Diagram

**Game Server**

* All user inputs via the menu go here.
* Controls what information is returned to the user.

**User External Devices** such as

* Keyboard
* Mouse
* Monitor

**Game Browser**

* Stores the MainMenu class.
* Returns user inputs to the server.

The application will work directly with the game server. The main thing the user will be seeing however is the game client, as that is where the menu and all related things will be displayed. When the user gives an input via the menu it is returned to the game server via the game browser, and then the next menu is displayed to the user via the game client again.

This application has no main user in mind, as it will be suitable for all ages; however, the predominant age of those who play these kinds of games range between 12-25 due to the rising popularity of Battle Royale games amongst the younger groups.

Section 2.2: User Stories

|  |  |
| --- | --- |
| UC1.1 | User opens app and enters login info |
| Description | User login screen |
| Rationale | User must have an account to use the application |
| Preconditions | None |
| Basic Course of Events | * User will enter an email address * User will enter a password * User will submit information * Server will authenticate information and either grant or deny access. |
| Postconditions | The user will be forwarded to the home menu screen |

|  |  |
| --- | --- |
| UC1.2 | User selects game start |
| Description | User attempts to start game |
| Rationale | User needs to go through this menu to start a game. |
| Preconditions | Must already be logged in. |
| Basic Course of Events | * User selects game start. * User is brought into the game customization screen. |
| Postconditions | User will have to select the settings for the game they want  to play. |

|  |  |
| --- | --- |
| UC2.0 | User Moving Camera |
| Description | User will be able to move in the game via a display on their phone, or with their mouse. |
| Rationale | In these types of games movement is key, so there must be convenient  means of moving around the map. |
| Preconditions | Must already be in the game. |
| Basic Course of Events | * User moves mouse in direction they would like their camera   to shift towards, and the camera follows the mouse.   * Mobile users will have a toggle they can move around in order to shift their camera angle. |
| Postconditions | The user will be able to navigate through the map. |

|  |  |
| --- | --- |
| UC2.1 | User Attacking. |
| Description | User will be able to attack other users in the game via left/right clicks  on their mouse, or with select buttons on mobile devices. |
| Rationale | User must be able to attack other characters or there would be no win  condition. |
| Preconditions | Must already be in game. |
| Basic Course of Events | * Once the user is already facing another character (already having aimed at them) they will be able to left click/right click and deploy   an attack towards that respective character. The user’s attack will vary depending on what kind of weapon they have equipped (sword, gun, etc) |
| Postconditions | The user will be able to attack other characters. |

|  |  |
| --- | --- |
| UC2.2 | Taking Damage |
| Description | Just like how the user can attack other characters, other characters will also be able to attack them. With being attacked comes taking damage, so this is what qualifies as a hit. |
| Rationale | User must be able to attack other characters or there would be no win  condition. |
| Preconditions | Must already be in game. |
| Basic Course of Events | * Once the user launches a specified attack (left/right click) against another character, the attack will obviously be deployed. Once that attack is deployed during its animation it will have a specified hitbox, and if that hitbox makes contact with another character’s avatar (their character model), then that respective character will take a pre-determined amount of damage (more for critical hits which are just hits in harder to hit specific locations) |
| Postconditions | Characters will be able to take damage, eventually dying or killing everyone else, thus losing or winning. |

|  |  |
| --- | --- |
| UC2.3 | User Dying |
| Description | Once the user takes a specific amount of damage(not accounting for healing, which will be included in the game) their character will “die” which lets the user start a new game. |
| Rationale | Game must end eventually, and this is it. |
| Preconditions | Must already be in game, and have your HP(Hit Points) drop to 0. |
| Basic Course of Events | * The user will be notified with an animation very time they take damage, and their damage taken will also be reflected on their health bar. * Once their health bar drops to 0, they are displayed with the game over screen and allowed to start a new game. |
| Postconditions | The user will be able to start another game in the case that they lose. |

|  |  |
| --- | --- |
| UC2.4 | Picking up items |
| Description | User will be able to walk over to items on the map and pick them up. These items both spawn randomly and are dropped from other characters if they’re killed. The more items a character had when they’re killed the more they’ll drop. |
| Rationale | Must have new items spawn in the game otherwise it would be very bland. |
| Preconditions | Must already be in game. |
| Basic Course of Events | * Random items spawn throughout the map and the user can walk up to them and pick them up via a key binding. * When a user is killed, where they died all their items will drop. |
| Postconditions | The user will be able to pick up items. |

|  |  |
| --- | --- |
| UC2.5 | User Movement |
| Description | User will be able to move their character via a toggle or their keyboard WASD keys. |
| Rationale | User must be able to move to navigate through the map. |
| Preconditions | Must already be in game. |
| Basic Course of Events | * The user will be able to move through the map using their W key to move forward, A key to move left, D key to move right and S key to move backwards. * Mobile users will have an on screen toggle they can flick to move. |
| Postconditions | The user will be able to navigate through the map. |

Section 2.3 External Interfaces

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Input # and name | Source | Description | Units of  measure | Data format | References to user stories, etc. |
| I1- Keyboard | Keyboard | User inputs commands/letters/numbers via keyboard buttons | Key clicks of the alphabet | Letters and numbers | UC 1.0  UC 2.4-2.5 |
| I2-Mouse | Mouse | Mouse cursor for menu traversal. Moves in game character | DPI (Dots per inch) | N/a | US1.1  UC 2.0-2.1 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Output # and name | Destination | Valid range, accuracy, and/or tolerance | Units of measure | Data formats | References to user stories, etc. |
| O1- Sound | Our game will have audible sounds and music | N/a | Decibels | Frequency | US1  US2  US3 |
| O2- Visuals | Our game will have visuals displayed on the screen | N/a | Brightness and color saturation | N/a | US1  US2  US3 |

Section 2.4: Non-Functional Requirements

2.4.1

* System will support maximum of 5 users playing simultaneously, not counting the AI that the users can include in their games.
* Almost no delay between user inputs and system processing due to it being a game. (No lag!)
* Will be able to store achievements that respective users have earned (achievements will be linked to user accounts, thus giving incentive to register)

2.4.2

* System will only access information on a basis of it being needed and at no other time to keep memory usage down.
* User information will be accessed if the user info tab is clicked on the menu, and no other information needs to be accessed.

2.4.3

* Game size is undecided but will not be over 1GB likely.
* Game will run on a minimum of integrated graphics but will run better with every increase in graphical power taken. (eg: Running the game with a GeForce GTX 760 which is a low-end graphics card will run better than Integrated Graphics)
* Game will need 1 GB of ram minimum to run safely but should not need more memory than that throughout the entire time the user plays the game.

2.4.4

* Database will only be accessible to the local machine, with certain credentials.
* User information will be stored on external machine and then ported into the game.

2.4.5

* All user data is backed up regularly
* User data will not be lost in the case of system failure as it will be backed up on external drives
* Game should be up 99% of time, assuming the server it’s hosted on doesn’t have any downtime.

2.4.6

* User data is backed up on external drives so user information will be safe in the case of a crash.
* We will work on having more than one server, such as a backup server in case the main server faces unexpected downtime.

**Chapter 3: Engineering Standards and Constraints**

Section 3.1: Engineering Standards

* All data will be encrypted via https with access being only granted to those with very specific credentials.
* Communication with other devices will be done strictly on an encrypted p2p(person to person) network.
* All data will be encrypted using symmetric encryption so that no one can access the information without a confidential key.
* All data will be stored on encrypted drives, and then backed up on external hard drives such that in the case of attacks/ransomware/natural server disasters.

Section 3.2: Realistic Constraints

1. Economical: No cost constraint. All we need is a server to host the game on.
2. Environmental: No major environmental constraints. Program is purely virtual.
3. Health and Safety: Game will display message to user every few hours to warn them about possible damage to their eyes from prolonged playing.
4. Ethical (Moral): All user data that is stored will not be given out to any third-party companies, and we will not collect any data from users greater than what we need for registration, as well as occasional data collections that we would use to improve the game, that users may opt out of.
5. Legal: All graphics and sound used will be in compliance with all laws, and no code or graphic will be taken from an outside source without either express permission, or having it already being open code.
6. Social: Game must appeal to the younger audience (12-25) which are the primary customers of Battle Royale games.
7. Political: No major political constraints. Game will not be politically controversial at all.
8. Manufacturing: No manufacturing constraints. Game is purely virtual.
9. Sustainability: System will be able to be recoded in order to add new “patches” which is just new features in the game potentially, new maps, new modes, things of that sort.

**Chapter 4: Project Design**

Section 4.1 Overview of System Components

**Database**

* Where the userInformation class is stored which contains all unique information about a user’s respective character.
* Will also store the AI behavior patterns.

**Database**

* Where the userInformation class is stored which contains all unique information about a user’s respective character.
* Will also store the AI behavior patterns.

**Game Client**

* Main thing the user will be seeing.
* All aspects relating to the game will be displayed through here.

**Game Server**

* All user inputs via the menu go here.
* Controls what information is returned to the user.

**User External Devices** such as

* Keyboard
* Mouse
* Monitor

**Game Browser**

* Stores the MainMenu class.
* Returns user inputs to the server.

Section 4.2 Structure and Relationship

1. UserInfo class is in charge of storing users login information. This class is called upon during log in and whenever a users data needs to be displayed
2. PlayerController class stores interactions and information between a player object and its surroundings
3. Camera class stores where the camera object is and what it displays at all times during the game
4. Unity will have all of our scripts and classes attached to game objects and will run on startup
5. Unity will have our map objects(trees, ground, terrain, water, and other physical objects) stored in the unity file
6. Unity scripts will include keyboard and mouse support attached to the correct game objects(player and menu).

Section 4.3 User Interfaces

1. U.I-1

User (Who): First Time User

Description (Does What): User will have downloaded the game and have the necessary tools to communicate with it (keyboard and mouse on PC, just a phone on mobile)

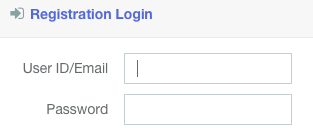
Reason (and Why): The user will do this initially so that they may register for the game.

Step-by-step description:

1.)Open app

2.)Select the “register” option

3.) Input ID/PW/Email and then submit it



1. U.I-2

User (Who): Someone who’s already registered for the game.

Description (Does What): User boots the game, and after the animations the game will present the user with a menu asking what they’d like to do, with options such as view their achievements, play the game, and exit the game.

Reason (and Why): User needs a menu to navigate through the game.

Step-by-step description:

-User Opens Game

-Game shows animations and then displays menu

-User has option to pick any choice,



1. U.I-3

User (Who)- Someone who’s viewing the main menu

Description (Does What): User presses the play option

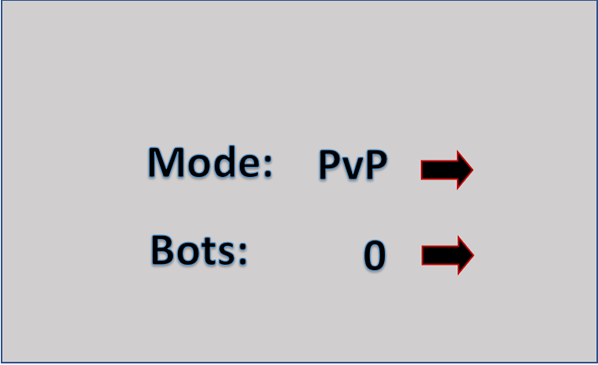
Reason (and Why): User begins playing the game by clicking on this option and going through the options

Step-by-step description:

-Menu pops up asking if the user would like to play PvP(Player vs Player), PvE(Player vs Environment).

-After picking, the user is asked if they would like to include AI(Artificial Intelligence) in the game. The AI in the game would basically fill the roll of other players in the game in the case that they wanted to play alone.

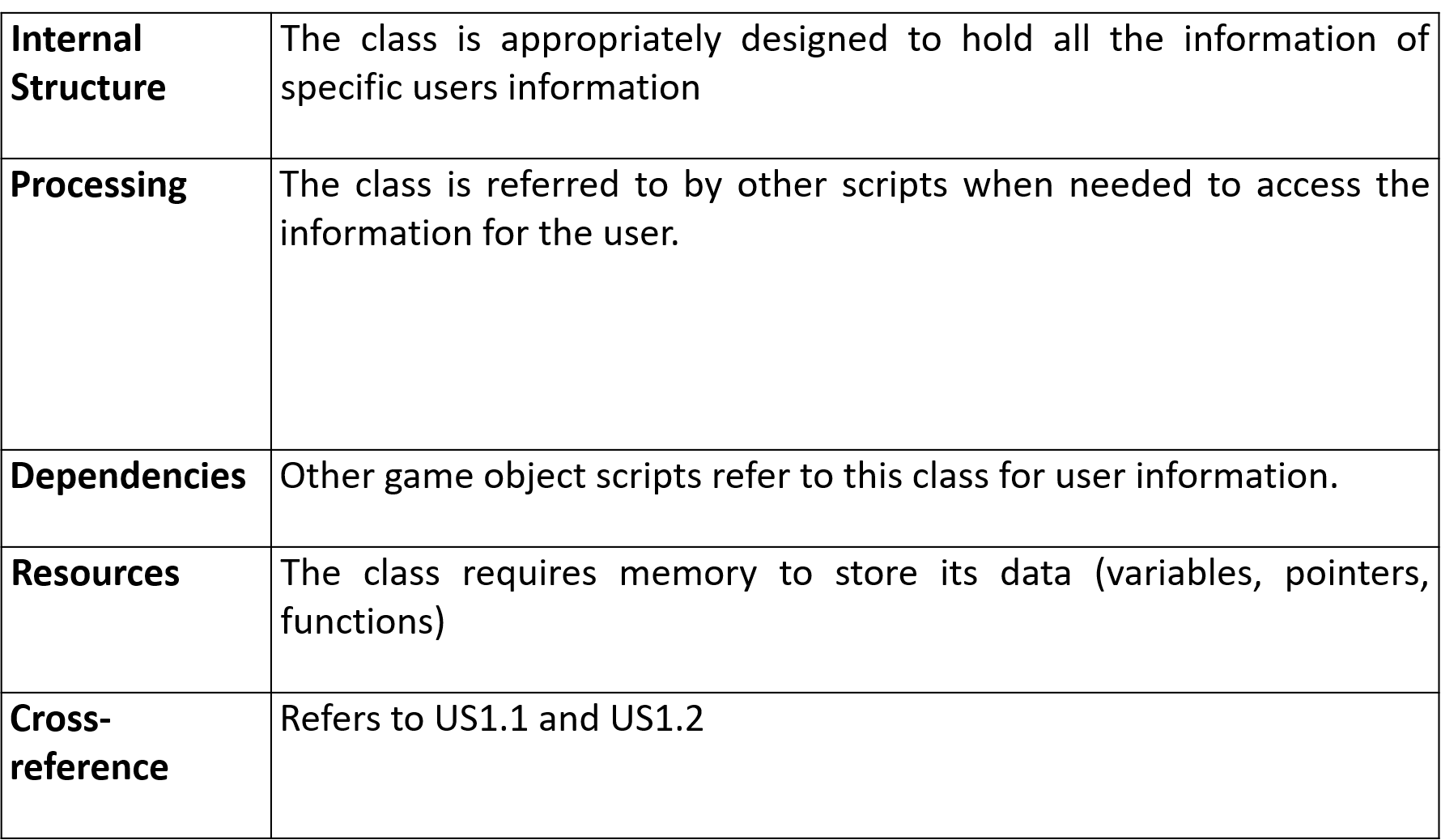
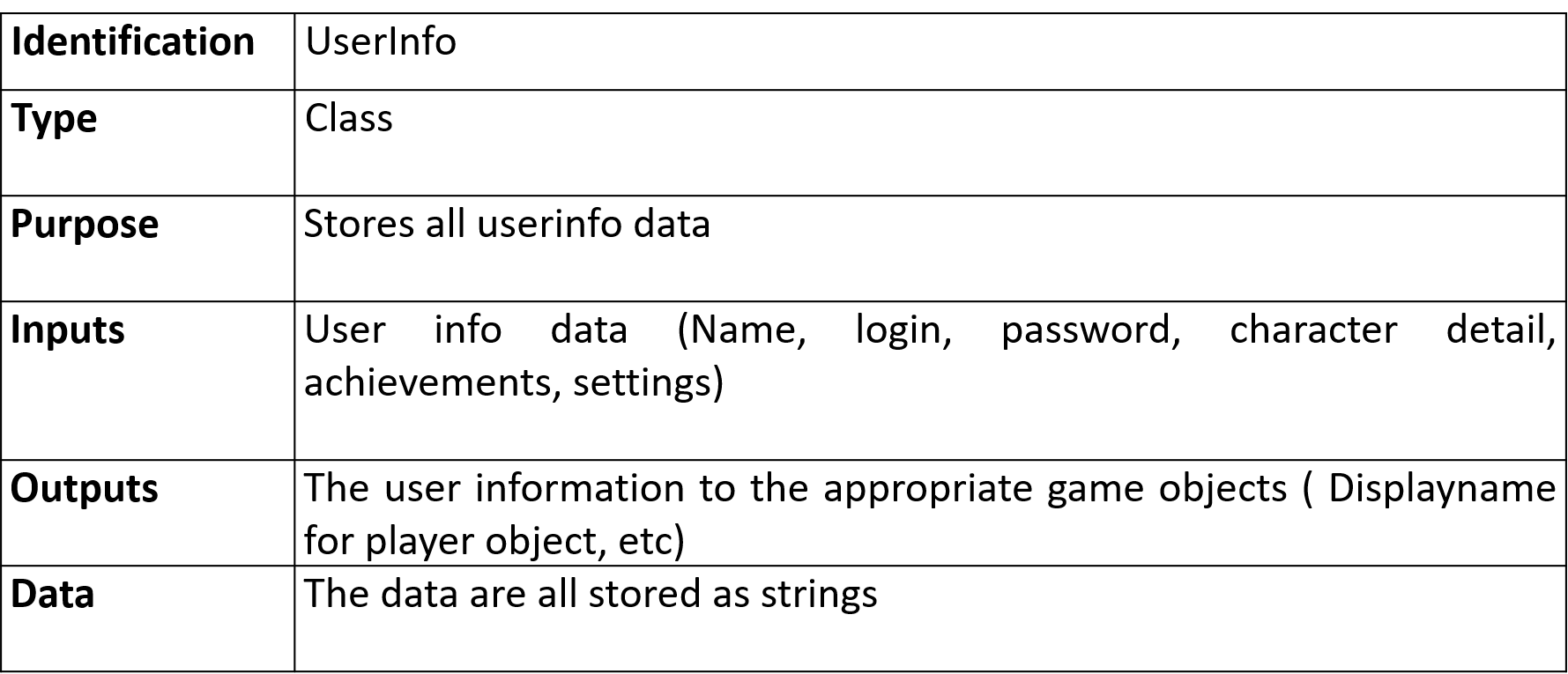
-User starts the game and the game load.



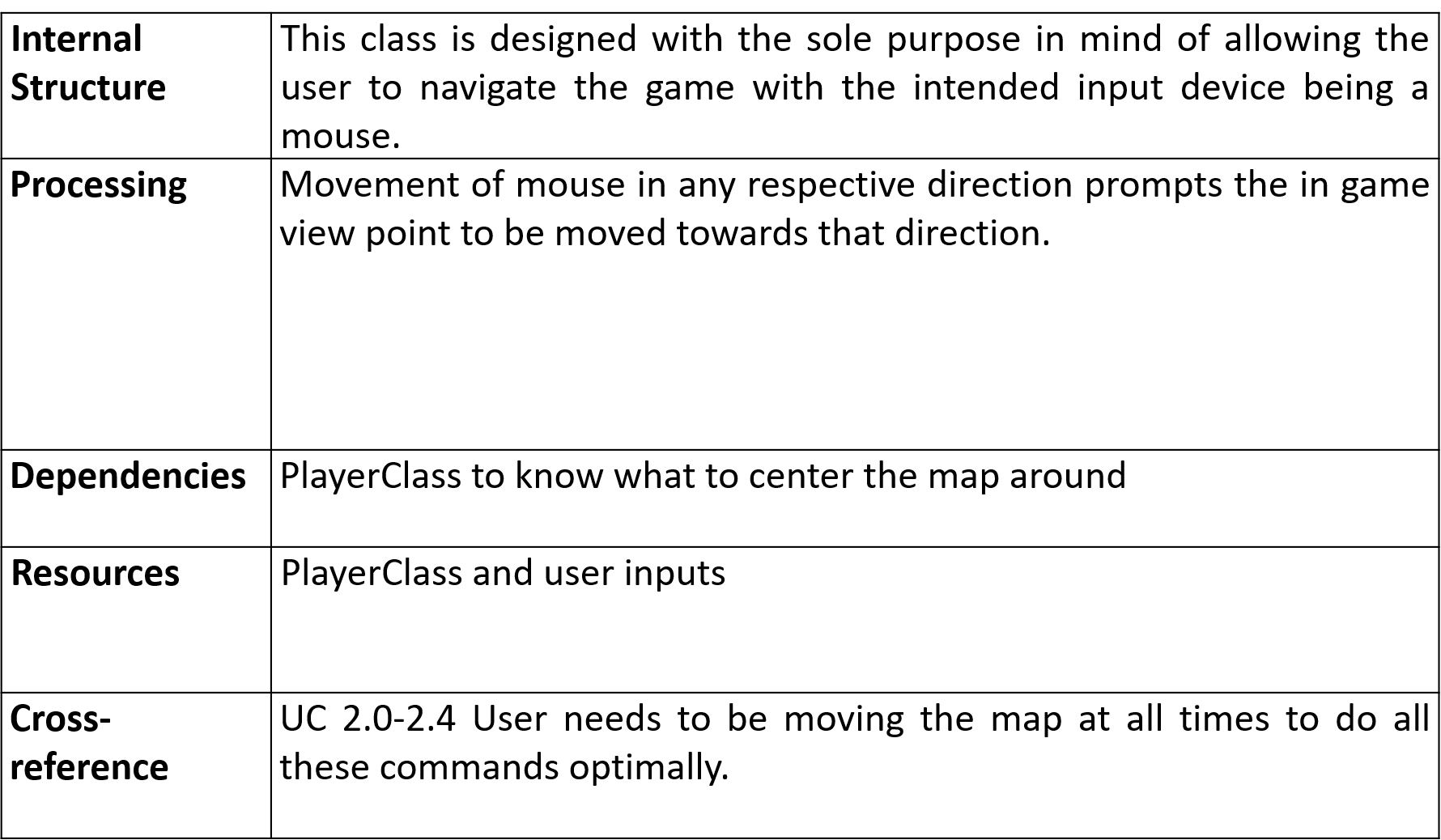
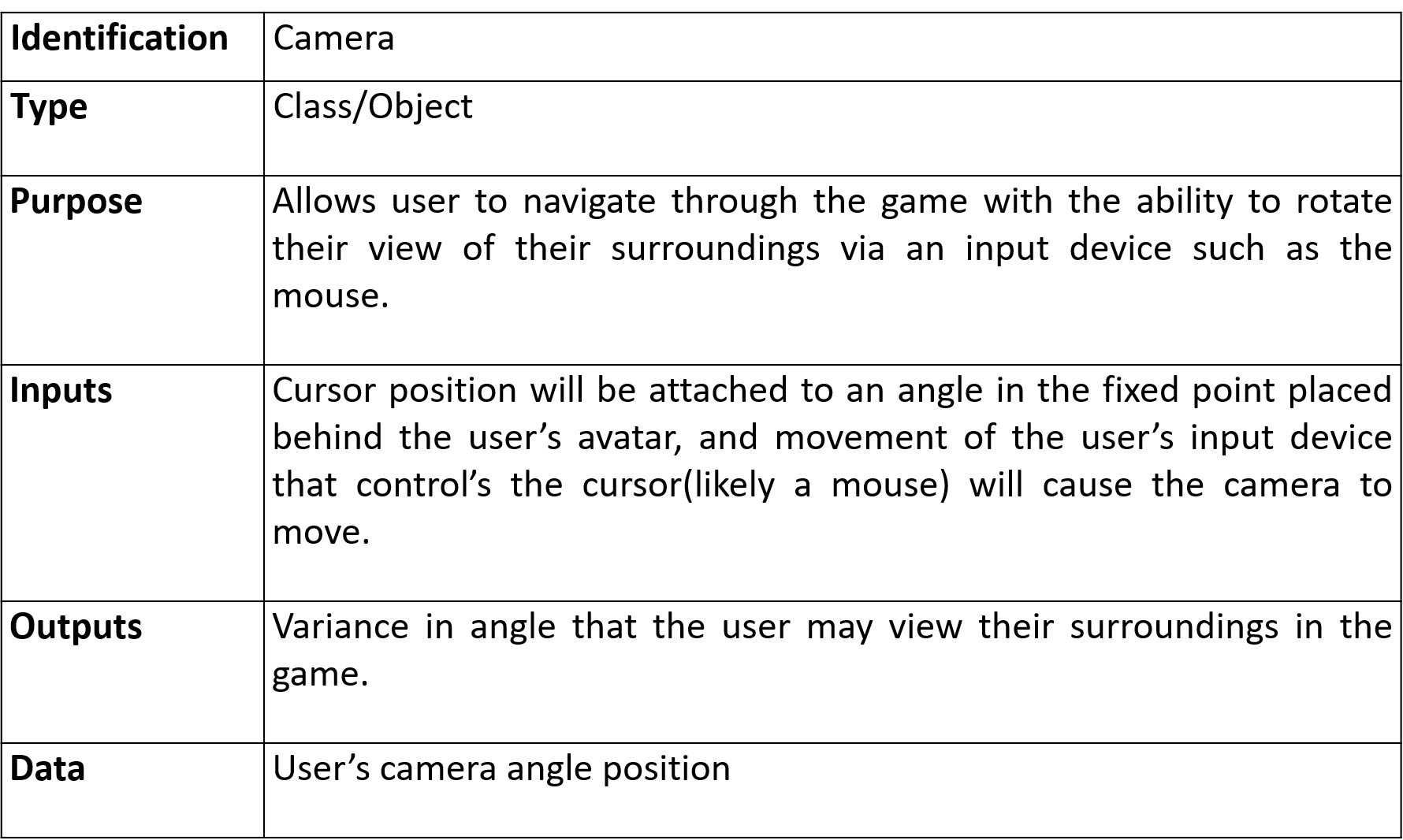
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **UI #** | **UI Name** | **Description** | **References with user stories, reqs, constraints** | **Dependencies**  **with other UIs** |
| **UI-1** | **Registration** | **User can register with email and PW** | **Requires valid email** |  |
| **UI-2** | **Main Menu** | **User can choose options (play, settings, etc)** | **No requirements** |  |
| **UI-3** | **Play Mode** | **What game mode to choose(PvE, PvP)** | **No requirements** | **UI-2** |

Section 4.4 Detailed Component Description

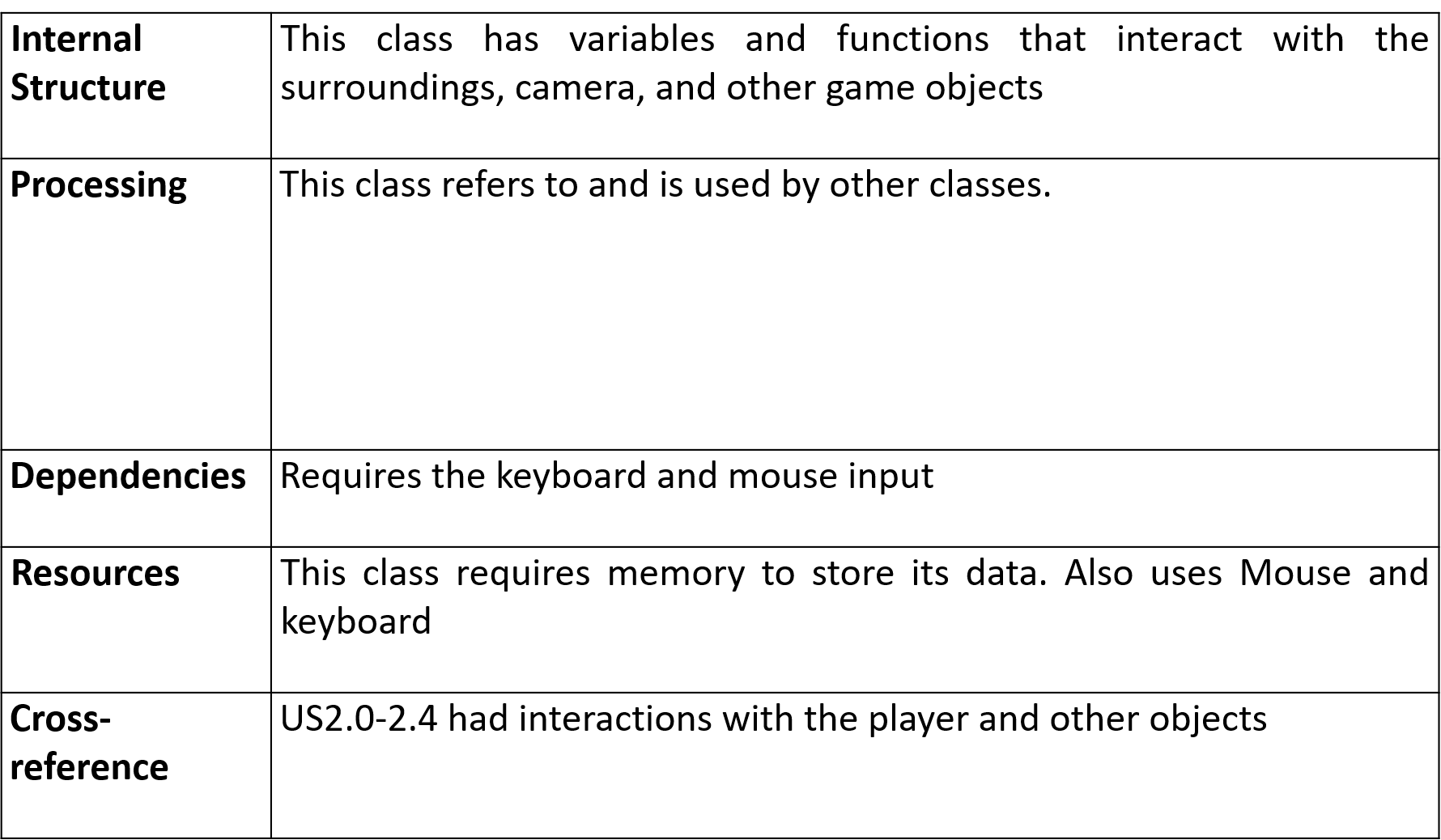
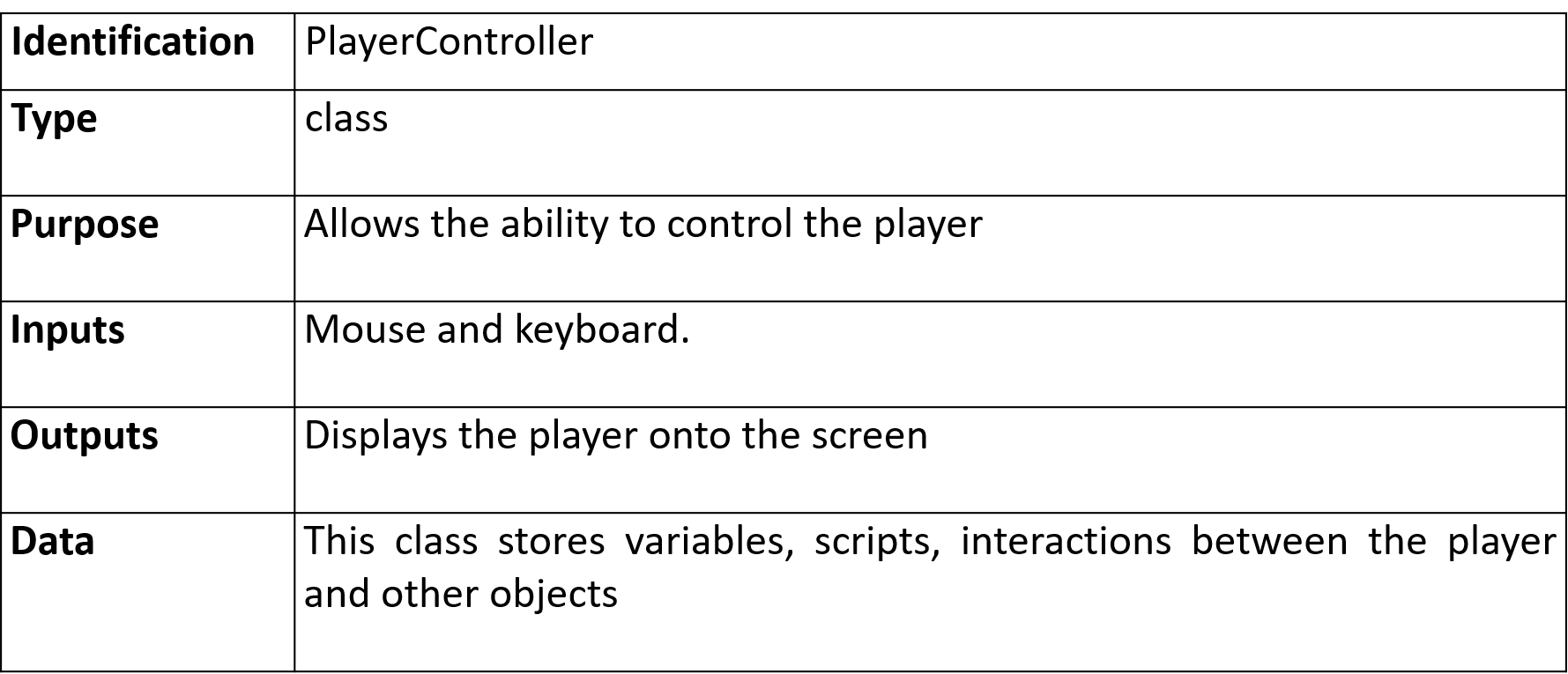
Section 4.4.1 UserInfo Class



Section 4.4.2 Camera Class



Section 4.4.3 PlayerController class



Section 4.5 Methodologies

With a small game like ours, using the newest version of Unity 3D will suffice in programming our game. Unity 3D comes with all the necessary tools and libraries to make a game. Our scripts and basic AI will be coded in C#.

Section 4.6 Design Alternatives

Our biggest decision was to choose what genre our game would be. We considered simpler options such as a 2D platformer but ultimately went with the more advanced third person shooting genre. Shooting games are much more popular in the world but much harder to make than 2D platformer game such as Flappy bird.

Section 4.7 Reuse and relationships with other products

Our game will be made from scratch. We will not use any code from other projects.

Section 4.8. Resource list

* Programming PC: personal pc capable of running Unity 3D; no extra costs
* Mina and Will: Game developers that will make the game; 5 hours a week per person
* Game server: Rented server to run our multiplayer function; 20$ per month

Section 4.9 Resource Skill List

|  |  |  |  |
| --- | --- | --- | --- |
| **Resource**  **name** | **Skill description** | **Where the skill will be applied** | **Reference to**  **design components**  **user stories** |
| Software engineering | Project design and testing | Cooperating with a team member | All user stories |
| Game Dev | Ability to design a game | Our entire project | All user stories |
| C++ | Learned the C++ language | Writing scripts | All user stories |
| SPM | Ability to manage software projects | Managing project | All user stories |

**Chapter 5: Project Plan**

Section 5.1 Sprints

5.1.1 Sprint 1

* **Goal**: Have a functioning basic version of our game
* **References**: US3
* **Starting** **date**: 1/1/19 **Duration**: 2 months
* **Sprint master**: Will
* **Demo goal**: A running game with movement, camera control, and guns

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Task**  **#** | **Task description** | **Hardware/Software resources** | **People(s)** | **Estimated hours** | **Points**  **difficulty** | **Cross-reference**  **user stories, require-ements, design** |
| 1 | Implement movement | Unity | 1 | 1 | 1 | n/a |
| 2 | Implement camera control | Unity | 1 | 1 | 2 | n/a |
| 3 | Implement guns and shooting | Unity | 2 | 3 | 4 | Design guns |
| 4 | Implement targets and Health | Unity | 2 | 6 | 6 | Design health UI |
| 5 | Implement game map and UI | Unity | 2 | 8 | 7 | n/a |

5.1.2 Sprint 2

* **Goal**: Implement basic AI into our game
* **References**: US3
* **Starting** **date**: 3/1/19 **Duration**: 1 months
* **Sprint master**: Mina
* **Demo goal**: Playing against bots that will shoot at other players

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Task**  **#** | **Task description** | **Hardware/Software resources** | **People(s)** | **Estimated hours** | **Points**  **difficulty** | **Cross-reference**  **user stories, require-ements, design** |
| 1 | Implement bot that shoots | Unity | 1 | 5 | 7 | Working bot |
| 2 | Varying difficulties of bots | Unity | 1 | 10 | 9 | n/a |

5.1.3 Sprint 3

* **Goal**: Finalize our game
* **References**: US1, US2, US3
* **Starting** **date**: 4/1/19 **Duration**: 2 months
* **Sprint master**: Will
* **Demo goal**: Our completed game with UI, AI, and multiplayer

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Task**  **#** | **Task description** | **Hardware/Software resources** | **People(s)** | **Estimated hours** | **Points**  **difficulty** | **Cross-reference**  **user stories, require-ements, design** |
| 1 | Finalize PvE mode | Unity | 2 | 6 | 6 | n/a |
| 2 | Implement multiplayer | Unity | 2 | 10 | 7 | n/a |
| 3 | Finishing touches | Unity | 2 | 3 | 2 | n/a |

Section 5.2 Risk Management

1. iOS game development might not be fully supported by Unity and its libraries. If this is the case, we have to code our own iOS game libraries. This risk is unlikely.
2. We may run out of time to fully design our game with all of our intended features. This risk is likely. We plan to add features as time goes by so we have a working game no matter what.

Section 5.3 Estimated Financial Budget

|  |  |
| --- | --- |
| **Item** | **Estimated Cost** |
| Tuition x2 | $80,000 |
| Unity Assets | $50 |
| Game Server | $20 |
| Total cost | $80,070.00 |

Section 5.5 Teamwork Plan

The team members for this game development project will meet weekly to stay on track and discuss the game. Jira will be used as an online management tool to manage tasks and team members roles/assignments.

**Chapter 6**

Section 6.1: Technical Methods

**PlayerShoot and WeaponManager** scripts both control everything related to the weapon. WeaponManager focuses on the stats of the individual weapon, it’s animations, the actual gun sprite, while PlayerShoot controls how the gun deals damage to other players.

**Player** script controls the stats of the player, their health, ability to fly, the player sprite, as well as the player’s ability to respawn.

**PlayerSetup** script controls the individual scene cameras as well as initializes all components needed for each individual player to host/join rooms on multiplayer mode.

**PlayerController** script controls player movement, as well as camera controls. Takes input from external devices such as:

* Keyboard
* Mouse

**Player** script controls the stats of the player, their health, ability to fly, the player sprite, as well as the player’s ability to respawn.

Appendix

All of these scripts connect to the unity multiplayer server uNet to communicate what each player is doing, merge it on their end, and allow for the communication between said players.

**PlayerController** script controls player movement, as well as camera controls. Takes input from external devices such as:

* Keyboard
* Mouse

**PlayerSetup** script controls the individual scene cameras as well as initializes all components needed for each individual player to host/join rooms on multiplayer mode.

**\*It is important to note that in Unity the scripts aren’t entirely what the game runs on, and almost, if not all of these scripts have to be tied in somehow to a Unity model for them to function properly. They do not function as standalones.**

**Description of what all the scripts do:**

**Player –** The Player script is meant to be utilized in cohesion with all the other Player prefix scripts. In this specific script the player is discerned between being alive or not, has their max health and current health instantiated, and tells Unity what prefabs to display while the character is alive, and what to remove from view when the character is dead. This script also has the code for the player respawn feature, which allows the player to come back after being defeated once. This Script is attached to the Player model in Unity.

**PlayerSetup –** The PlayerSetup script initializes all the prefabs that go along with the player model, sets up the playerUI, discerns if the player is local or playing from a different machine. It does all of these checks on the initial run of the client, and applies these settings by passing all these checks onto the Unity Player model that this script is tied to.

**PlayerMotor -**  This script controls all movement related things to the character, as well as everything tied to camera rotation. This does this by relating our character’s 3D model to their vector location on the map. This script is also tied into the Player model.

**PlayerController –** This script controls all joint motion for the character, allowing the characters “limbs” to move freely. This script also controls the character’s ability to fly, initializing the “thruster” settings to control how fast the character will fly.

**PlayerShoot –** This script tags each player in order to differentiate them in the server, and assigns the player a weapon. The script also initializes the reload function, as well as allowing the player to shoot the assigned weapon. This script also invokes all the muzzle flash animations and all other shooting animations.

**WeaponManager –** This script instantiates settings for all available guns, how much damage the gun does, the rate of fire, and has several checks to stop the game from glitching(not allowing the player to spam reload before the animation concludes, assigning a gun in the start of the game, etc).

**PlayerWeapon –** This script controls how much damage each respective weapon does, the max amount of bullets in a gun as well as how long it takes to reload the gun.

**CursorHide -**  This script turns off the cursor so that it isn’t visible while the game is being played, but when the pause menu is active the cursor becomes visible again.

**GameManager –** This script assigns the camera to each respective player so that when multiple players are in the game each player’s camera is attached to their own respective model.

**HostGame** – This script connects to Unity’s uNet server and is attached to the Create Room button, as well as instantiating the room name off of the user input. It does this through Unity’s built in StartMatchMaker function.

**JoinGame –** This script contains all the functions to deal with joining a room. It reads available room names from the MatchMaker function, has a refresh function to read these room names, reads the amount of players in any current room and outputs it through the UI.

**MatchSettings -**  This script is very simple, all it does is initialize the respawn time for each player and is used as a necessary placeholder to be tied into the pre-existing Player model.

**PauseMenu -**  This script has controls the pause menu, and utilizes Unity’s built in DropConnection function in order to allow the player to leave any room that they’re in, and takes them back to the main menu.

Section 6.2 Tools

* We made our project using Unity 3D game creator.
* IDE used alongside unity was Microsoft Visual Studio 2017, in which we coded all our scripts in C#.

Section 6.3 Programming Languages:

* Every script we wrote was written in C#.

Section 6.4 Use of Open Source Code:

* We used several assets from the Unity Playstore as it would simply be too time consuming to animate those ourselves, all of which were free to use.

Section 6.5 New Tools and Knowledge

* Unity 3D: Unity 3D is a IDE that has several built in animations, the full support of it’s playstore and is extremely well documented. Unity also has a built in multiplayer hosting service, and is extremely user friendly, especially to new users. All of these factors combined made it an extremely easy choice to work in.
* Visual Studio 2017: Microsoft Visual Studio is Microsoft’s semi-universal IDE which we used to write all of our scripts. Visual Studio is also extremely user friendly, not only tracking errors as soon as they’re written, but also suggests fixes for it.

**Project Implementation and Results**

Section 7.1 Implementation Strategy:

Our team agreed to split the work halfway, as neither of us had really any experience using Unity prior to doing this project. We both had an interest in game development, so we just decided to assign tasks to ourselves and essentially split the project down the middle. Through the help of several tutorials and many long hours of Unity practice, the game eventually came to fruition.

Section 7.2 Sprints:

In the first sprint we made a base game for us to work off of, and built ontop of as it would make no sense to immediately dive into features without a base for our project. We had to do things such as character model creation, map creation, weapon creation and making the basis for all of our scripts. This arguably was the hardest sprint as not only were we learning Unity, but we also were brushing up on C#. Once the scripts were put together we found out the sad truth that coding would be the easier part of this project, as the rest of the work had to be done using Unity’s built in tools. Unity is a marvelous tool, assuming you know how to use it, but we didn’t and as such it was very frustrating trying to figure out character physics, movement, why we kept falling through the map and things of that sort. Things that we thought would be elementary were turning out to be extremely frustrating. We got through this, and moved onto the second sprint. In the second sprint unfortunately our frustrations continued as we learned about the difficulties of incorporating multiplayer. We struggled for most of the duration of this sprint until we found out at the end how to implement uNet to work with our game, and everything was smooth sailing from there. We tested several connections, added all the expected features from this sprint and moved onto sprint 3. In sprint 3 we tried to add our AI(term used loosely) features to the game. This is where the real issues came in, as at this point we were experienced in Unity but still couldn’t figure out why the scripted AI we were using wouldn’t behave properly in our multiplayer server. We would later find out that scripted AIs don’t particularly cooperate with online multiplayer through Unity and that we’d have to use a different kind of method of botting if we wanted to incorporate this feature. Unfortunately due to time constraints we couldn’t do this, so we had to drop this feature.

**Sprint 1 Backlog**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Task Desc** | **Hours Expected** | **Actual Hours** | **Start Date** | **Finish Date** | **Completion %** |
| Create Character Model and basic animations | 5h | 8h | 1/13/19 | 1/13/19 | 100% |
| Create (initial) Map | 3h | 3h | 1/15/19 | 1/15/19 | 100% |
| Assign Character Physics | 5h | 9h | 1/20/19 | 1/24/19 | 100% |
| Create weapons class and make weapons | 6h | 10h | 1/25/19 | 1/29/19 | 100% |
| Assign shooting function to player | 2h | 2h | 2/3/19 | 2/3/19 | 100% |
| Assign health, and death function to player | 5h | 5h | 2/7/19 | 2/8/19 | 100% |
| Create player respawn | 6h | 6h | 2/11/19 | 2/14/19 | 100% |

**Sprint 2 Backlog**

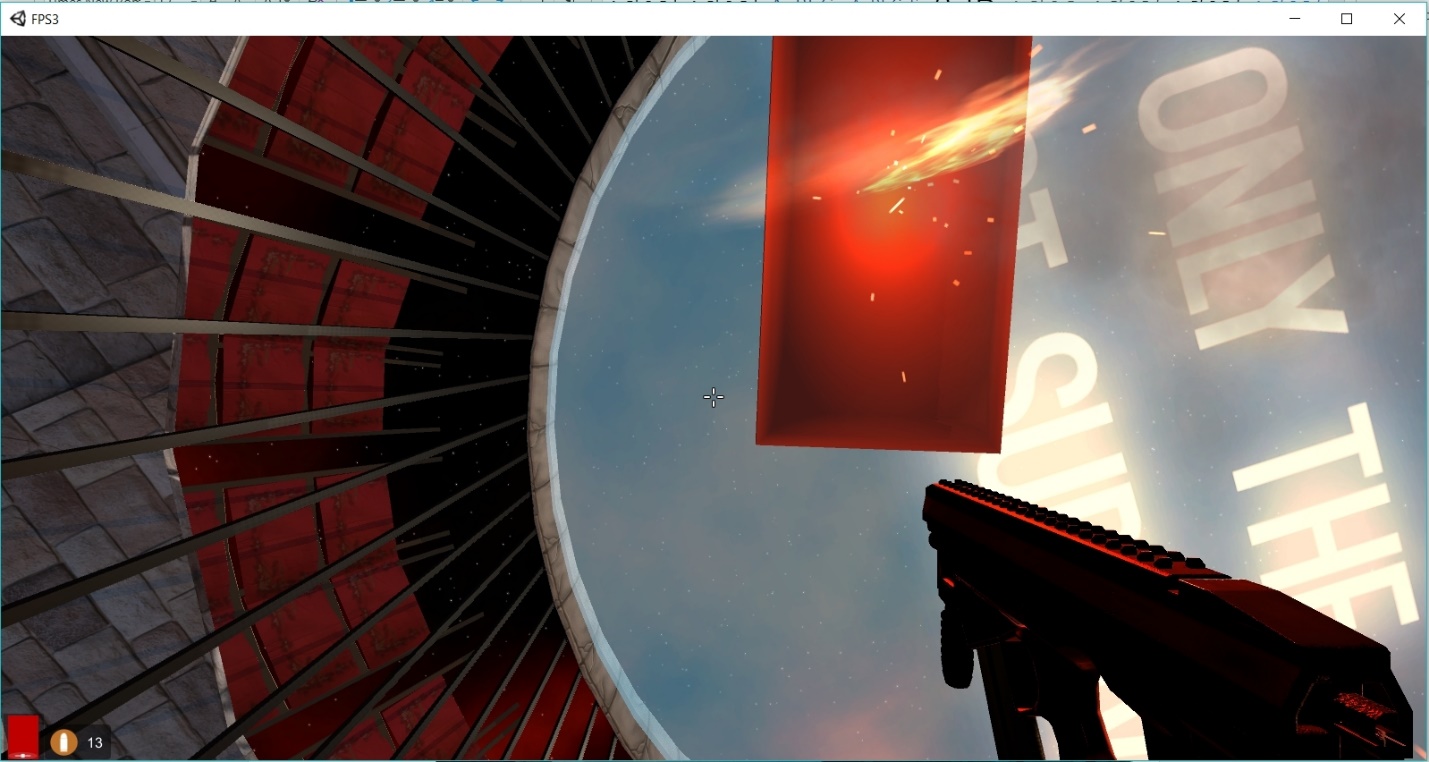
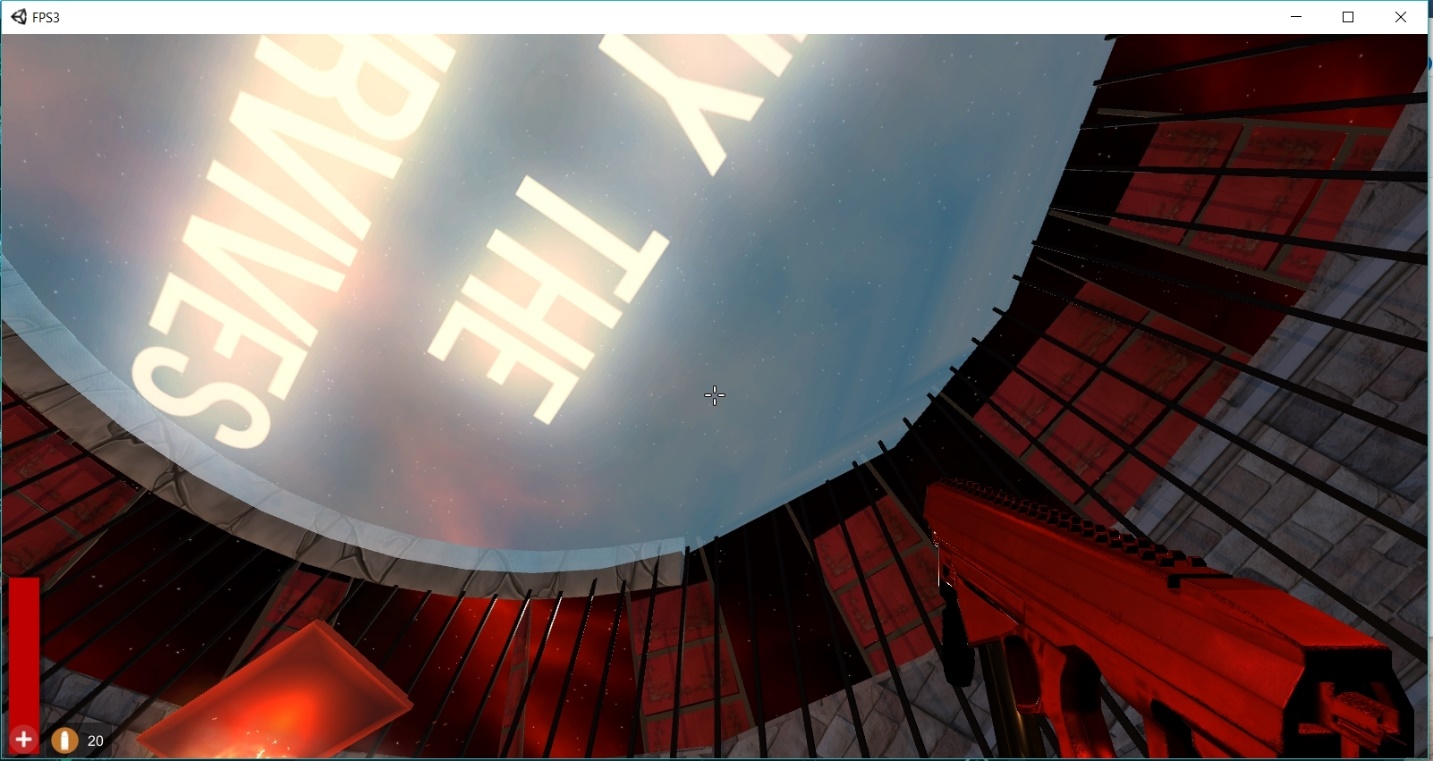
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Task** | **Hours Expected** | **Actual Hours** | **Start Date** | **Finish Date** | **Completion %** |
| uNet implementation | 6h | 12h | 3/7/19 | 3/13/19 | 100% |
| New Homepage UI | 6h | 5h | 3/16/19 | 3/17/19 | 100% |
| Integration of uNet functionality onto the home UI | 4h | 4h | 3/17/19 | 3/19/19 | 100% |
| Connecting all the scripts so that they could communicate with uNet as opposed to just being client sided | 6h | 8h | 3/26/19 | 3/29/19 | 100% |
| Adding a disconnect button and ensuring the user could connect to any room they wanted | 3h | 5h | 4/4/19 | 4/6/19 | 100% |

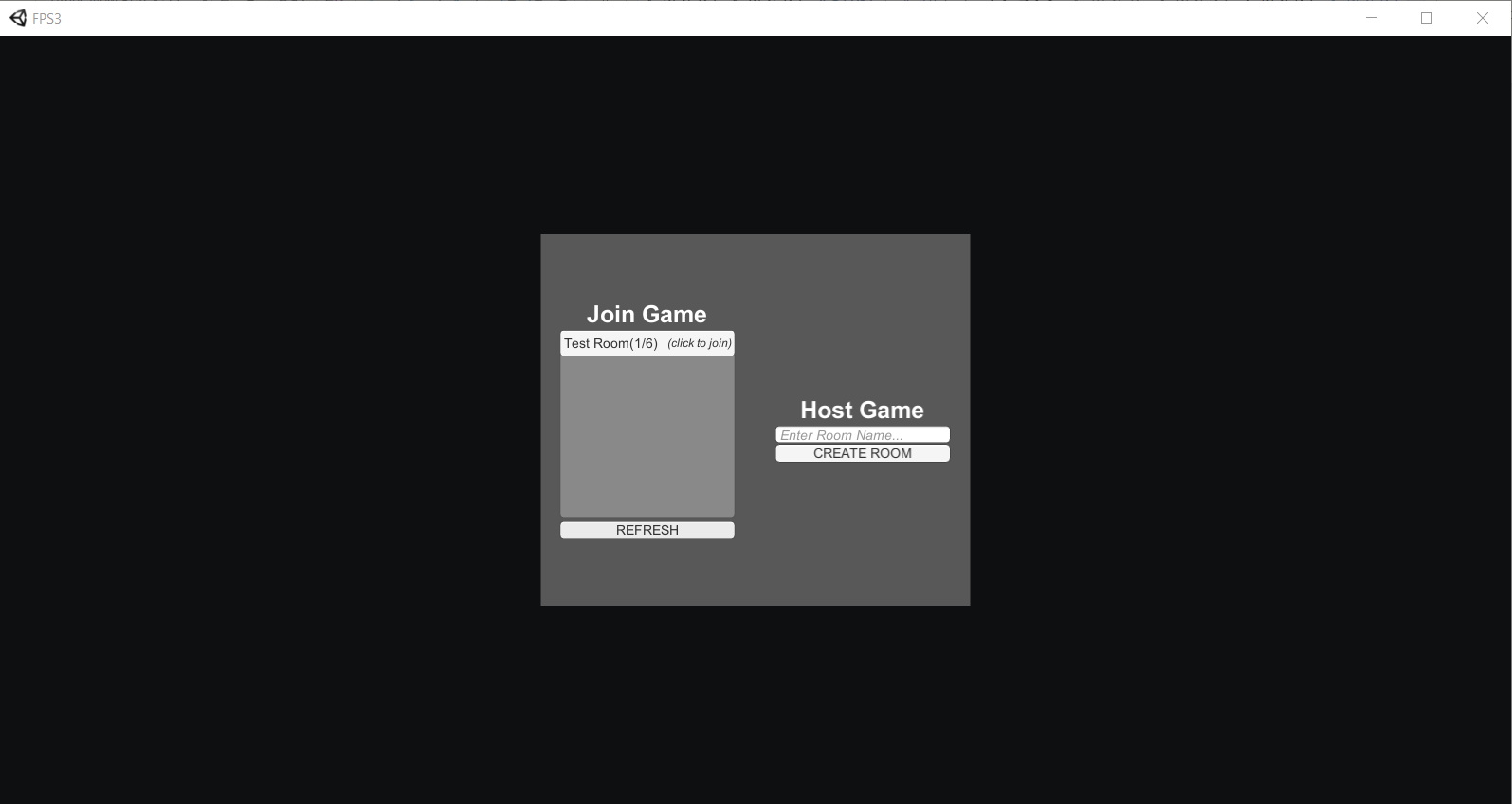
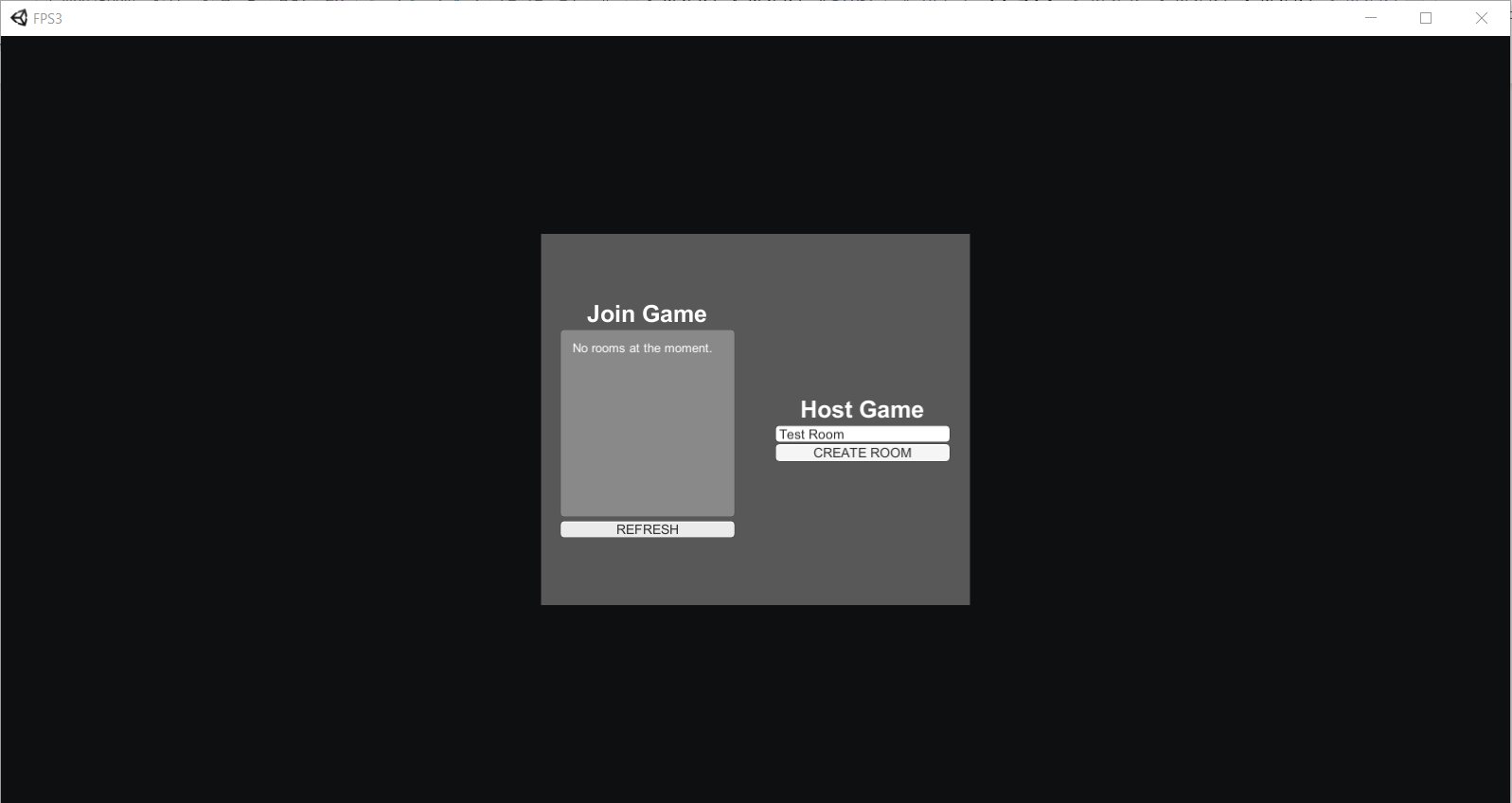
**Sprint 3 Backlog**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Task** | **Hours Expected** | **Actual Hours** | **Start Date** | **Finish Date** | **Completion %** |
| **Finalized Map Creation** | **3h** | **3.5h** | **4/24/19** | **4/24/19** | **100%** |
| **Added animations to new map** | **3h** | **2h** | **4/25/19** | **4/27/19** | **100%** |
| **Added reload animations/ability to reload** | **3h** | **4h** | **4/30/19** | **5/2/19** | **100%** |
| **Added visible health bars** | **2h** | **2h** | **5/3/19** | **5/4/19** | **100%** |
| **Attempted to add a working AI(bot) system into our multiplayer mode** | **7h** | **XX** | **5/4/19** | **XX** | **50%** |

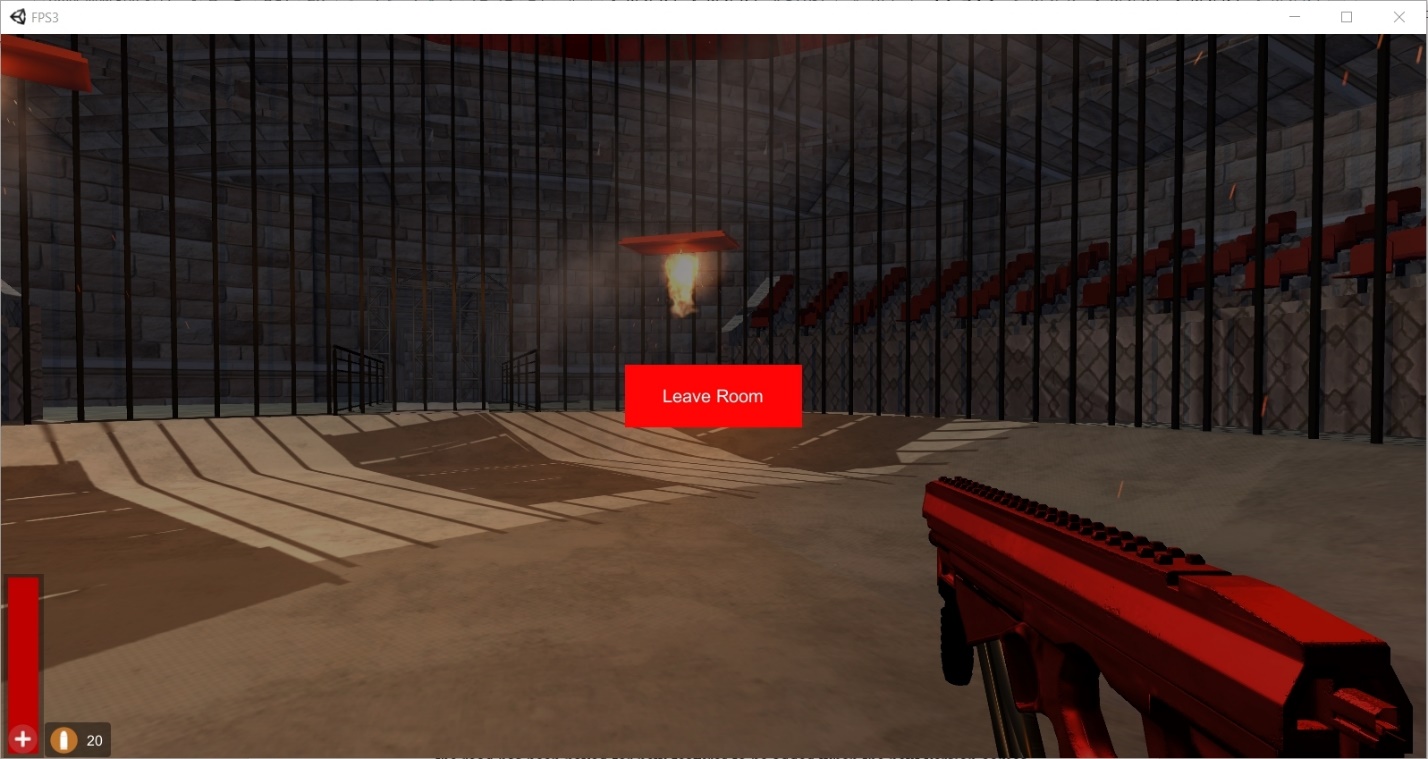
**Unfortunately we had to remove the AI system from this version of the game due to imposed time constraints. That is why the last requirement in sprint 3 wasn’t completed.**

Section 7.3 Results

* Nuclear Winter does have a fully functional base, including character, proper physics, and several (animated) features such as flying, reloading, bullet animations, and gun animations.
* As you can see in this screenshot it shows the user having full health and 20 ammo. After some combat you see in the second screenshot the character is wounded(lower health bar) as well as having less ammo. 
* It is Multiplayer and will support upto 20 simultaneous players(which can be upgraded for a monthly fee).



As you can see the room name that will be created is called Test Room, and when the room is made it’s visible on another client.

* Users are allowed to host their own rooms, join other user rooms, and leave and join new rooms as they please with no need to reboot the game. 
* There is no lag(latency) issues in the multiplayer rooms, and those rooms can be hosted over the internet, as opposed to our first sprint in which the rooms had to be hosted over LAN.
* Game has no critical flaws and users can play as long as they want with no memory leakage issues.
* While we didn’t finish the game due to time constraints, it is fully maintainable and the road has been paved for new features to be added when the next version comes.

**Chapter 8: Problems and Changes**

When making our application we ran into an unexpected issue while attempting to implement our AI(scripted) into our multiplayer mode. While implementing a client side scripted AI is possible, integration of a scripted AI into Unity’s multiplayer system uNet was incredibly glitchy and required special integration that we just didn’t have time to work around due to imposed time constraints. Unfortunately we had no valid solution for this due to this project being much more challenging than anticipated, so we just had to drop it.

**Appendix**

