**Design Document I**

**<** Space Crucible**>**

**REVISION HISTORY**

|  |  |  |  |
| --- | --- | --- | --- |
| Revision # | Author | Revision Date | Comments |
| 1.0 | Parth Patel | September 24, 2021 | initiated, add sequence diagram for use case 2 |
| 1.1 | Parth Patel | September 26, 2021 | Add server class diagram and description, server component description |
| 1.2 | Isaac Colon,  Meshwa Patel,  Yifan Zhang,  Kwadwo Gyasi-Danquah  Parth Patel | September 26, 2021 | Add state diagrams, use case sequence diagrams, game object class diagram, client application component desc |
| 1.3 | Parth Patel | November 22, 2021 | Updated system overview  Added new use cases, sequence diagram, state diagrams, and class diagrams. Revised component descriptions |
|  |  |  |  |

**Table of Contents**

[System Overview 4](#_Toc88569333)

[General Requirements 6](#_Toc88569334)

[Component Descriptions and Interfaces 6](#_Toc88569335)

[Class Diagrams for Components 8](#_Toc88569336)

[Use Case Sequence Diagram 11](#_Toc88569337)

[Use Case #1: 11](#_Toc88569338)

[Use Case #2: 12](#_Toc88569339)

[Use Case #3: 14](#_Toc88569340)

[Use Case #4: 15](#_Toc88569341)

[Use Case #5: 16](#_Toc88569342)

[Use Case #6: 17](#_Toc88569343)

[Use Case #7: 18](#_Toc88569344)

[Use Case #8: 19](#_Toc88569345)

[State Diagram 20](#_Toc88569346)

## System Overview

Space Crucible is a two-dimensional, top-down perspective action-puzzle game with support for both single and multiple players. Levels will be defined as a series of square tiles on a fixed grid- however, player and monster movement will not be locked to discrete points on this grid. The combat will be in real-time and take place on tile-based levels. Space Crucible will have a science fiction theme and will emphasize run-and-gun combat style. Players take on the role of “Exterminators” and must use a variety of tactics to combat foes, from weaponry to coercing them to fight each other.

The objective is to explore the map while eliminating monsters, dodging traps, and finding the exit to proceed to the next level. Multiplayer is a key feature; levels can be designed that require more than one player to complete (in fact, the multiplayer-oriented level design will be prioritized). A level editor will be included so users can create their scenarios and save them in a simple text-based format. Levels can be compiled into level packs using a simple archive format called a “.WAD”, which contains and organizes any necessary graphics, sounds, music, and monster code (defined in scripts, rather than hard-coded Java).

Players will receive a launcher upon starting the game. The launcher will allow the user to pick a level pack, a specific level from the pack, choose a difficulty, and either launch a single-player session or join a multiplayer session by inputting the lobby code in the join lobby menu. Real-time action will use WASD or arrow keys to move the player, while the player can simultaneously aim using the mouse. Certain map tiles or objects can be interacted with to proceed in the level or trigger traps. An in-game chat will allow players to communicate with each other. A straightforward light system will allow tiles far from light “sources” to darken, obscuring important puzzle components or hiding sneak attacks, including bridges that the players can pass over or under. MIDI files will be used as background music.

There will be a Master server that will handle all clients connecting to the multiplayer game mode. Once a player has selected create a lobby or join a lobby option, they will be directed to a specific game server. Each lobby will have a game server that will handle all incoming connections and data from the clients. In multiplayer mode, the game server controls the whole game to ensure that all players are in sync and the host does not have an advantage over other players. The clients are only responsible for sending their input data to the game server; the game server handles moving the players and sending the updated rendering data back to the clients.

Players will be able to host their own lobbies by selecting Create Lobby option in the co-op mode menu. Each lobby will have its unique 4-digit code that the host can share with other players to invite them to the lobby. The 4-digit code is linked with the lobby’s IP address and port number. When the client makes a request to create a lobby, the master server sends the lobby info to the client, and then the client joins the lobby. The players who want to join an existing lobby will choose the Join Lobby option and input the unique 4-digit code. The join lobby request is sent to the master server, and the master server finds the lobby associated with the code and sends the lobby details back to the client. The lobby host will have access to the difficulty selection menu and start game option that other players won’t be able to see. Once the host selects the start game option, the level beings for all players in the lobby. If the host leaves the lobby before starting a game, the host privileges are passed on to the player who joined second.

Once the lobby host starts the level, new players can still join the ongoing level as long as they have the lobby code. This feature also allows players to leave and join the ongoing game as they wish. Another feature we implemented is if a player leaves in an ongoing game, the player will be replaced by a bot/AI player who will help the players finish the level. The bot player can follow the closet player and shoot and chase the visible enemies.

Server administrators can use the remote control (RCON) application to monitor the servers and send instructions to perform certain tasks. The RCON client can log into both the master server and the game servers using a lobby code (or MASTER for the master server) and a password (the master server’s host chooses the master password, the game server password is generated randomly upon lobby creation). When RCON is connected to the master server, it can monitor the number of game servers the master is hosting and check their lobby codes, RCON passwords, and add-on files. When connected to a game server, player positions and health and packet information can be monitored in real-time, chat can be sent to the players from the server, and settings such as game skill and level can be changed.

The program will be written in Java and will support all desktop operating systems (i.e., Windows, macOS, and Linux). There will be separate executable JAR files for the master and game servers, client programs.

A Java IDE will be necessary to develop this project, preferably a common one shared by all developers. The Java LibGDX library is a game development library that includes graphic, sound, music, and networking functionality and will be the framework we use to develop the game. An open-source library, DoomStruct, exists to manipulate .WAD files, which will be used to access game data neatly in an archive format. The KryoNet library is used for network operations, exchanging packets between the game client and the master and game servers over the TCP protocol.

## General Requirements

* Desktop or Laptop running Windows, Mac, or Linux operating systems
* Master and Game Servers will run on a Linux machine (headless, desktop or laptop)
* Java runtime environment
* Keyboard and mouse to control in-game movement
* Optional - Internet connection to access multiplayer mode

## Component Descriptions and Interfaces

Space Crucibles comprises of three high-level components: client application, master server, and game server.

**Client Application:** The game features two modes, single-player and multiplayer. In the single-player mode, the client application by itself is enough to play the game. In multiplayer mode, the user running the game will need an internet connection to connect to both the master server and game server to play the game.

In the single-player mode, the client application controls the game logic, renders the game data, and handles client input. This way, the client can play the game without having access to an internet connection.

In the co-op mode, the multiplayer feature, the client application will be responsible for connecting and sending input data to the servers and rendering the data that the server sends back.

When the player selects the co-op option in the main menu, they will be able to join/create a lobby where they will wait until the lobby host starts the game. If the user creates a lobby, they will become the lobby host and have access to start the game and change difficulty options. Once the host starts the game, all players will spawn and move using WASD and aim and shoot using the mouse and left click, respectively. The player can attack the AI monster until its health goes down to zero and move forward. If the player’s health reduces to zero, the player sets to the beginning of the level. The player can clear the level once they have passed all the obstacles and reached the endpoint. In the settings, the player can control the sound and audio. If required, they can turn it down to zero or adjust it accordingly. The game will also have a text chat where the players will communicate with each other. After the game starts, the AI monsters, controlled by the server, will attack the player while they are finishing the level. WASD or arrow keys are used to move the character. The player can aim using the mouse towards the AI monster.

Users can create their level packs using a level editor. Users' new levels will be compiled into level packs using an archive format- ‘.WAD’. To play the user-created levels, they can load them using the addons option inside the settings menu. Once the level is loaded, the user can either go into single-player and play the game or go to multiplayer and play the level. When the invited players try to join the lobby with a custom level loaded, the server automatically downloads the level files from the host and loads them to allow the user to enter the lobby.

**Master Server:** The master server will be hosted on a Linux machine. We will be using KryoNet, a Java networking library. It will provide efficient TCP client/server network communication using NIO. There will be only one instance of the master server running at all times. The master server will be responsible for keeping track of all game servers and redirecting all clients to the game servers. When a game server is no longer being used, it will ping the master server, and the master server will be able to reuse it to create another lobby.

**Game Server:** The game servers will also be hosted on a Linux machine, and they will be created using the Kryonet library. We will create as many instances of the game servers as needed. The game server will control the game logic, keep all players in sync, handle input data by updating the game, and send the render data back to all clients. The game server will be the brains of the game in multiplayer and control everything to ensure everything stays in sync and fair for all players. The Game Logic runs at approximately 55 ticks per second. At each tick, we will send the player’s x and y coordinates to the server and update the timer if the player has moved.

## Class Diagrams for Components

**A screenshot of a computer program

Description automatically generated with low confidence**

**Figure 1**: Master Server Class Diagram

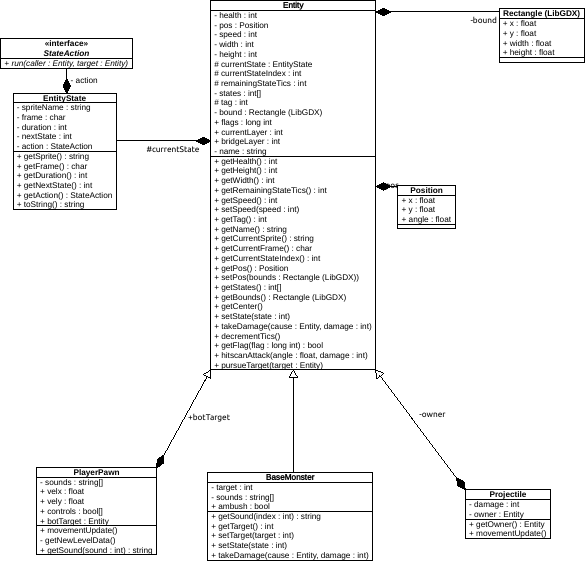
**Figure 1**shows how the MasterServer class communicates with the clients. The MasterServer is a class that represents a server; in Kryonet, a server has a threaded listener, and the listener has its own connected, received, and disconnected methods, so whenever a client connects, sends data, or disconnects, the listener handles it. So, when the user wants to access coop mode, a new SpaceClient is created, and that class tries to connect to the MasterServer. If the SpaceClient is able to connect, then the client can access the co-op mode menu; otherwise, an error is shown to the client. The MasterServer class also has a client/server relationship with the game server, where the master server plays the role of server, and the game server plays the role of a client. Whenever a new SpaceServer connects to the MasterServer, it records the id and port number in a HashMap. So, whenever a client requests to create a lobby, the MasterServer checks if a SpaceServer is available to be used. If one is found, the MasterServer sends the server details to the client. When a client tries to join a lobby using a lobby code, the master server checks if that lobby code corresponds to any active SpaceServer, and if it does, the MasterServer sends the details of that SpaceServer to the client. When all players leave the SpaceServer, the SpaceServer pings the MasterServer that it’s empty, so the MasterServer can make it available to use again.

A screenshot of a computer program

Description automatically generated with low confidence

**Figure 2**: SpaceServer Class Diagram

**Figure 2**shows how the SpaceServer class coordinates with the client and game logic to run the game. SpaceServer has two roles: a server that SpaceClients connect to and a client that connects to the MasterServer. When clients send requests to create or join a lobby, the MasterServer sends the details to connect to a SpaceServer. When the SpaceClient connects to the SpaceServer, they change the screen from TitleScreen to LobbyScreen, where they can see all of the players that have joined so far. When the lobby host starts the game, SpaceClient sends a ping to the SpaceServer to start the GameLogic loop and spawn all the players in the game. On the client-side, they switch from the LobbyScreen to the GameScreen. The GameLogic is controlled only by the SpaceServer, so when clients send input to the SpaceServer, it updates the player’s coordinates and sends the render data back to the SpaceClient for rendering. This cycle continues until the game is over or all players leave the server. Once all players have left the SpaceServer, it sends a ping to the master server that it's empty and ready to be reused.



**Figure 3**: Game Object Class Diagram

**Figure 3** shows the basic game object class hierarchy. An “Entity” is any map object that represents an active creature or person. These are split mostly between monsters, which will inherit from an abstract “BaseMonster” class, and players, who are represented by the PlayerPawn class. Both contain basic information such as health, position, speed, size, what graphics they use, and what angle they are facing. The abstract BaseMonster class defines methods which any monsters in-game will need to override in order to have functionality- for example, idle() represents an enemy waiting to see players, chase() is called when the monsters are pursuing their target, and so on. Target is a pointer to an Entity existing on the map that the monster is chasing. This may not necessarily be a player. The PlayerPawn class contains information such as which player client the entity is representing (via playerNumber), what keys, ammo, and weapons are currently in the player’s possession, and the methods called when the PlayerPawn is moving, using something on the map, attacking, or dying. Other classes include Item, the base class for anything that can be picked up by players, and projectiles, which contain a pointer to whatever Entity launched it (for example, so a player can take the credit for a kill).

## Use Case Sequence Diagram

### Use Case #1:

Harold wants to spend more time with his grandson David, who is very much into the shooter genre. Harold wants to play with his grandson, so he creates a lobby and invites him:

1. From the main menu Harold clicks “Co-op” option and is presented with two options “Create lobby” and “Join lobby”.
2. Harold clicks Create lobby option to host his own lobby. When he loads into the lobby, he sees a lobby code in the bottom left corner of the screen. Harold shares his lobby code to David.
3. To join his grandfather’s lobby, David clicks on the “Join lobby” option. He is prompted with a text box where he can input the lobby code, David inputs the lobby code and clicks “Ok”.
4. David loads into the lobby and both Harold and David are able to see each other’s usernames in the lobby. Since Harold is the host of the lobby, he has access to the start game option. Harold clicks start game and two players load into the game.

Table

Description automatically generated with medium confidence

**Figure** **3a**: Sequence Diagram for Use Case 1

### Use Case #2:

Alex just downloaded the game and wants to play the game in single player mode.

1. Alex boots up the game and loads into the main menu
2. Alex selects single player on the main menu
3. Alex selects “very easy” option in the difficulty window and clicks “Ok” to confirm
4. Alex’s player entity loads into the game
5. Alex presses W, A, S, and D on his keyboard to control the up, left, right, and down movement of the player
6. Alex moves the mouse to aim his weapon and clicks left click to shoot his weapon
7. Alex explores the map and kills the monsters along the way.
8. Alex finds the exit and runs to it when he reached the exit, he spawns into the next level.

Diagram

Description automatically generated

**Figure 3b**: Sequence Diagram for Use Case 2

### Use Case #3:

David recently downloaded Space Crucible out of curiosity and blasted through the game with his friends. They enjoyed the game, but now they have completed all of the default levels and are quite bored. However, thanks to the game’s built-in level editor, David can resolve this dilemma by creating his own levels.

1. David opens the grid-based editor and first creates the shapes of the rooms in his new scenario.
2. Then, he adds the player start points, and any monsters, traps, or scenery in the level after that.
3. When he saves his custom level, the editor takes the new level info and stores it in a simple text-based format.
4. He can compile many of these level files in a .WAD archive, along with any new graphics or sounds he may need.
5. He can then distribute the .WAD to his friends and play his new scenario- or even post it on the internet for strangers to download and play.

A picture containing diagram

Description automatically generated

**Figure 3c**: Sequence diagram for Use Case 3

### Use Case #4:

Bob wants to change his in-game username.

1. Bob loads into the main menu.
2. Bob selects the Settings option on the main menu.
3. Bob clicks the “Change player name” option.
4. Bob enters “Bob” in the text field and clicks “Confirm”.
5. Now when Bob plays the game, he has a username on top of his player that says “Bob”.

Diagram

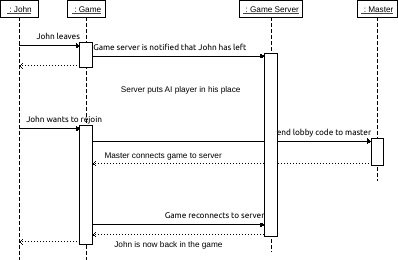
Description automatically generated

**Figure 3d**: Sequence diagram for Use Case 4

### Use Case #5:

John gets a phone call and has to leave the game. But after he is done, he wants to join back into the lobby.

1. John is playing a game in co-op mode with his friends.
2. John gets a phone call and has to attend the call.
3. John exits the game, and a bot player replaces him.
4. The bot player plays on John’s behalf.
5. John finishes the call and rejoins the game using the same lobby code.



**Figure 3e**: Sequence Diagram for Use Case 5

### Use Case #6:

James wants to mute the music in-game because he wants to listen to his own music.

1. James clicks on the “Settings” option on the main menu screen.
2. In the settings menu James selects “Volume” option.
3. Inside the volume menu James turns down the SFX slider to zero and BGM slider to zero.
4. Both game volume and music get muted so now James can listen to his own music.

**Diagram

Description automatically generated**

**Figure** **3f**: Sequence Diagram for Use Case 6

### Use Case #7:

Pedro’s friend created a new, challenging level and Pedro wants to help him test it.

1. Pedro downloads the .WAD file that his friend created
2. Pedro opens the Space Crucible client, goes to settings menu, and clicks addons option.
3. Pedro uses the built-in file browser that addons has and chooses the downloaded .WAD file.
4. He can now play single-player or multiplayer with the new levels, and any other additions.

**Diagram

Description automatically generated**

**Figure** **3g**: Sequence Diagram for Use Case 7

### Use Case #8:

Sam has cleared all the levels by himself in the easy mode. He wants more of a challenge, so he turns up the difficulty to the highest mode to try it.

1. Sam selects the single-player option on the main menu.
2. In the choose difficulty level menu he chooses the nightmare option which is the hardest difficulty in the game.
3. When Sam clicks go, he spawns into the game and is immediately ambushed by a monster.
4. Sam finds this mode to be very challenging and has fun trying to clear it.

**Diagram

Description automatically generated**

**Figure** **3h**: Sequence Diagram for Use Case 8

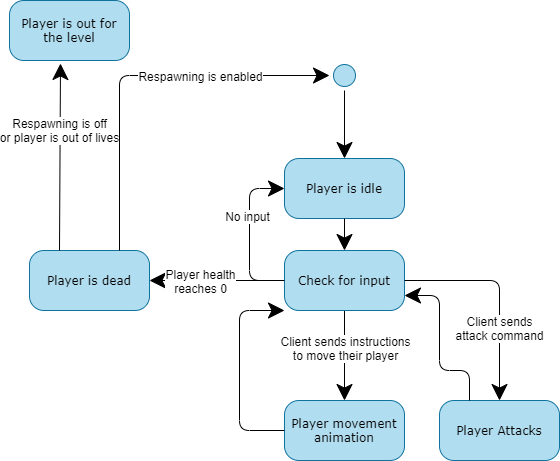
## State Diagram

Diagram

Description automatically generated

**Figure 4**: State Diagram for user

**Figure 4** is the state diagram for the users who wants to start a game. From the main menu, users have two options Single Player and CO-OP. If the user selects Single Player, they are shown the episode selection screen. After choosing an episode, the first level is loaded, and the player spawns in the game. After the player reaches the exit of all levels in the game, the episode is cleared, and the user is sent back to the main menu. If the user selects CO-OP, they are shown two options: to create their own lobby or join an existing lobby. Users who want to create their lobby are led to the lobby screen and set as the lobby host. Now they can invite their friends by sharing the lobby code. If the join lobby option is selected, they are shown a text box to input the lobby code. If the lobby code is correct, they will be led to the host’s lobby. The lobby host has permission to start the game for everyone, so they wait in the lobby screen until the host starts the game. The host starts the game, and an episode (a collection of levels that are played in a row) is chosen, and the first level of this episode is loaded. Player pawns are spawned in the level, and gameplay commences. Once the level is cleared, the server checks if there is another level to be loaded in the episode. If there is, players are taken directly to the next level. Only once all levels in an episode are beaten (unless the game ends prematurely) are the players then sent back to the lobby where they can choose a new episode or leave the game.



**Figure 5**: State Diagram for player

**Figure 5** is the player state loop. At the start of the level the player is spawned on its designated spawn point as defined by the map editor. Each player pawn (that is, a player as represented physically on the map) is controlled by one connected client. If the client is not inputting any commands, the player pawn remains idle. The client can command their avatar to move and shoot (or possibly use objects on the map such as switches). The player has a health property which is reduced when it takes damage and increased when it picks up health. If health reaches 0 or below, the player dies and is respawned at the start, unless respawning is off, or player is out of lives (if the server has declared a finite amount).

**Diagram

Description automatically generated**

**Figure 6**: State Diagram for Master Server

**Figure 6** shows how the master server communicates with all clients. The master server has a threaded listener that listens for all incoming connections, packets, and disconnections. As you can see in Figure 6, most of the work done by the master client is done in the received method, where the master client receives packets from the clients connected to it. When the master server receives a SendServerInfo packet, it stores packet data and makes it usable for clients. When the server receives create lobby packet, it looks into the list of available servers and sends the details back to the client. For the join lobby packet, the server uses the lobby code that the client sent to look up the game server associated with it and send back the details of the game server. For CreateWadFile and WadFile packets, the server redirects all data to the client that requested it. The RCON login and message are packets sent by the RCON, and the master server sends back the information it requested. The game server sends the open lobby packet when it is free and available to be reused.

Diagram

Description automatically generated

**Figure 7**: State Diagram for Game Server

**Figure 7**shows how the game server handles the clients and connects to the master server. On execution, the game server first connects to the master server, and when it’s connected, it sends a server info packet that helps the master server identify it as a server. On the other hand, the game server has another listener that handles all incoming connections, packets, and disconnections. When clients connect to the game server, they are stored in a list. When clients send packets, they are captured by the received method. The add tile, state, camera data, add object, and game entity packets are used to update the game logic. The input data represents the clients’ inputs, and the server directs it for the game logic to process, and the server sends the updated render data back to the clients. The send player name packet is used to share the names of all users with each other. The start game packet handles starting the game for all players waiting in the lobby. The RCON message packets are redirected to game logic to process the commands. The send ping packet is a way to check for the latency between the client and server. The check connection packet lets the server identify who sent the packet, a player or RCON user, and check to see if the game has been started, then spawn the player in the game directly or show the lobby screen.