**Test Procedures**

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

**REVISION HISTORY**

|  |  |  |  |
| --- | --- | --- | --- |
| Revision # | Author | Revision Date | Comments |
| 1.0 | Parth Patel | September 8, 2021 | initiated |
| 2.0 | Isaac Colon,  Parth Patel,  Meshwa Patel,  Yifan Zhang,  Kwadwo Gyasi-Danquah | September 9, 2021 | Added unit tests, integration tests, and acceptance tests |
| 3.0 | Kwadwo Gyasi-Danquah | November 22, 2021 | Initial revisions based on feedback |
| 3.1 | Parth Patel | November 23, 2021 | Update system overview |
| 3.2 | Meshwa Patel, Isaac Colon, Parth Patel | November 28, 2021 | Updated Client Unit Tests |

**Table of Contents**

[Document Overview 4](#_Toc89028907)

[System Overview 4](#_Toc89028908)

[Unit Tests 6](#_Toc89028909)

[Client Component 6](#_Toc89028910)

[Server Component 6](#_Toc89028911)

[Integration Tests 8](#_Toc89028912)

[Acceptance Tests 10](#_Toc89028913)

## Document Overview

The document will provide a high-level overview of the game as well as define the testing procedures that we will implement during our development phase. The document includes the following sections:

* System Overview
* Unit Tests
* Integration Tests
* Acceptance Tests

## 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. 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.

## Unit Tests

### Client Component

|  |  |  |
| --- | --- | --- |
| **Action Button** | | |
| Procedure | Automated unit tests for UI objects on the title and settings screens. For testing if a certain menu is open, we can call a visibility check on its associated Scene2D UI object. | |
| Test | Input | Output |
| Start | Click on the start button. | Checks if the player can successfully reach the next game screen where difficulty level of game is selected. |
| Difficulty levels | Test client selects a random difficulty level for single-player mode. | Checking if the difficulty level changes properly. It is relatively easy on very that other levels. |
| Co-op | Select the Co-op option on the main menu. | Checks if the player can successfully reach the next screen that asks the user to join a lobby or create a lobby |
| Join a Lobby | Select join lobby option. | Takes user to screen where lobby code is to be entered. |
| Create a lobby | Select create lobby option from the co-op menu. | Request is sent to the server , and it generated a code and takes the player into the lobby. Other players can join using the generated code. |
| Settings | Select settings from main menu. | Takes the user to the next screen where level add-ons and audio settings can be done. |
| Volume Slider | Set volume slider to zero. | The game volume should turn off and the sequencer should be muted |

### Server Component

|  |  |  |
| --- | --- | --- |
| **Client Lobby Connection** | | |
| Procedure | 1. Create a field where user can input lobby code.  2. Process the user input. | |
| Fields | Text input for lobby code | |
| Test | Input | Output |
| Lobby Does not Exists | Guest test client sends lobby code for invalid lobby. | Connection should be denied, since lobby is invalid. |
| Lobby Exists | Guest test client sends lobby code for valid lobby. | Check if the lobby exists or not. If the lobby exists connect the client to the lobby server. |

|  |  |  |
| --- | --- | --- |
| **Level Selected** | | |
| Procedure | Create a field where user can select a map and send the map selected to the server. | |
| Fields | Clickable objects for maps | |
| Test | Input | Output |
| Map Selected | Test client sends a level change packet to server. | Check if server received the map selected and check if player is being loaded into the map. |

|  |  |  |
| --- | --- | --- |
| **Select Difficulty Level** | | |
| Procedure | Create a field where user select the difficulty level | |
| Fields | A menu with clickable buttons | |
| Test | Input | Output |
| Difficulty selected | Test client sends difficulty change packet to server | Check if the difficulty level selected is being sent to the server. |

|  |  |  |
| --- | --- | --- |
| **Server Input Handling** | | |
| Procedure | 1. Receive client’s input from the keyboard | |
| Test | Input | Output |
| Player moves | Sends the boolean array controls to server. | Check if the server received the correct input from the client. |
| Player rotates the player entity | Sends the angle 180-degree angle to server | Check if the server received the correct angle the player is facing from the client. |
| Player shoots the weapon | Send the boolean array controls to server with shoot equals to true | Check if the server received the mouse input from the client. |

|  |  |  |
| --- | --- | --- |
| **Loading Player Entity** | | |
| Procedure | 1. Receive how many players are in a lobby 2. Create the player entities in the map | |
| Test | Input | Output |
| Player joins lobby | Receive number of players in the lobby | Check if the number of player entities loaded in map match the number of players in lobby. |
| Player leaves lobby/game | Receive number of players in the lobby | Check if the correct player entity is removed from the game. |

|  |  |  |
| --- | --- | --- |
| Test | Input | Output |
| Key bind tests | Position integer (x or y), Robot input to simulate key pressed | True if Robot input was recognized and position integer was changed accordingly |
| Show screen test | Screen, Boolean | True if screen in context is show, else false |
| Resize screen test | Screen, its dimensions | True if the height and/or width are different, else false |
| Pause screen test | Screen, its state | True if its state is paused, else false |
| Resume screen test | Screen, its state | Use robot to pause screen via key input, then use resume method. True if state is not paused, else false. |
| Dispose screen test | Screen, its resources | True if it has released all its resources, else false |

## Integration Tests

Integration testing will be handled by GitHub Actions. Since the program is a game which uses OpenGL via LibGDX, a separate headless LibGDX client application will need to be created for the purpose of performing integration tests (because the main client programs will not open in a headless environment).

1. WAD Functions Test:

The headless client will attempt to open the default .WAD file and retrieve level and Entity data. This is essential functionality for single-player and multi-player modes, as well as making add-on content with the editor.

Input: open a .WAD file into a WadFile object.

Expected result: retrieve usable LevelData and EntitySpawner objects stored in List structures.

1. Level Editor Test:

The headless program will contain methods to create LevelData structures, add new tiles or objects, and export them to files similar to the level editor (though without any OpenGL functionality that won’t run in a headless environment).

Input: a standalone sample add-on .WAD and a standalone sample .LMP (single-level file).

Expected result: program saves new, modified copies of the files, which are verifibly different from the originals, but still valid files.

1. Level Loading Test:

The headless client will attempt to load a level from the default .WAD and initialize the game loop in single player. Then, it will retry with a second test add-on .WAD.

Input: load from default resource.wad as well as a sample add-on .WAD.

Expected result: levels from both default and add-on .WADs are loaded, and game logic begins.

1. Server Test:

The server (which, being headless, will be able to run in an integration testing environment as-is) program will launch. Clients will connect to the server and wait in a lobby. When all clients are connected, end the lobby, and begin the game loop.

Expected result: game loop begins, and a multiplayer session commences.

## Acceptance Tests

Acceptance Test will be handled by developers. Since it is the last step of the testing phase, it should test all the features we have implemented so far.

|  |  |
| --- | --- |
| **Run the Space Crucible application.** For Windows, run SpaceCrucible.jar For Linux, run SpaceCrucible.sh For Mac, run LegacyCrucible.sh in terminal | The application should load up and the main menu should show |
| **Change username.** On the main menu, under the settings option, there will be another option to change the player’s name. | The player’s name from default, which is player(number), will be changed to what you type in and would show up in the game. |
| **Change Volume.** On the main menu, go to settings. There will be an option to change the volume. | The volume should toggle from 0 to 100, initially set to 50. |
| **Play Single Player Mode.** On the main menu, click on single-player option, a menu will pop-up for level difficulty, choose a difficulty from the dropdown menu and click ok to load the game**.** While in the game use WASD to move and use the mouse to aim and shoot. | After selecting the level of difficulty, the game should start in a single-player mode. |
| **Leave Single Player Mode.** While playing Single Player mode press "ESC" key to open pause menu and select Exit. Or if player dies and wants to leave, select Exit. | After selecting exit you should load back to the main menu. |
| **Select Co-op option** On the main menu, click on Co-op option | If the master server is running you will see the Coop mode menu. If the master server is not running, then you will see an appropriate error. |
| **Create a Lobby.** Under the co-op option, there are two options, one of which is Create a Lobby. Player selects this option if he wishes to host a lobby. | After a short pause, a new lobby will be created with a lobby code and an RCON password for the host. The host will be able to start the game whenever they want. If the host leaves, the next lowest player number becomes the host. If the lobby goes empty, it is removed. |
| **Join a Lobby.** Under the co-op mode in the main-menu, there are two options, one of which is joining a lobby. Select that option and enter the lobby code one wishes to join. | If that lobby code exists, the player will be able to get into the lobby and wait for the host to start the game. If the lobby does not exist, then an error message will appear saying that the lobby code does not exist. If each client's add-ons match the host's, they should be able to join, else they'll get an error that the file names/hashes don't match. |
| **Change Lobby Host.** When in a lobby with at least 2 players the player who created the lobby should select Exit Lobby option or close the game. | Once the host leaves, the player who joined second should be the new lobby host and he should have the start game button. |
| **Leave and rejoin the same lobby** Open two games if testing by yourself. Use one to create lobby and use another one to join the lobby that was created. Record the lobby code because you will need it later. Once both players have joined let host select start game. After loading into the level have any one player leave. After leaving, the player who left should try to join the same lobby again with the same lobby code. | When the player leaves a bot player should replace the player who left. Once the player who left joins the lobby again his player entity should spawn in and be able to play. |
| **Load Add-On.** From the settings menu, clicking the "Add-ons" button will open the users file directory. The user may then select an appropriate .WAD file to load additional assets stored within it into the game | If the loaded .WAD file contains assets recognizable to the game, the player should be able to access new levels, enemies, and features |
| **Edit Level** Go to the main menu and choose the "level editor". Open a .WAD (archive of many levels) or .LMP (single level in a standalone file). Right-click to add a tile or shift and right-click to add an Entity. Hold Ctrl and drag mouse to select many tiles, or Ctrl-Shift and drag to select many Entities. Middle click deletes tiles and Entities. Ctrl-C and Ctrl-V copy-paste tiles, and Shift Ctrl-C/V copy-pastes Entities. Ctrl-O opens a new file, and Ctrl-Shift-O opens a new level in the current file (if it is a .WAD). Ctrl-S saves. The map name and music track can be modified in the bottom left text fields. | Levels should be able to have their tiles and entities modified, loading new resources from add-on .WADs as necessary. Saving a .WAD should add or edit the current Level inside the .WAD while saving a .LMP should modify the single, standalone level file. |
| **Edit .WAD** Using the SLADE 3 editor (third-party), open a .WAD and edit its assets. Define a new entity using the ENTITIES lump and provide sprites by adding graphics between S\_START and S\_END marker lumps and sounds between FX\_START and FX\_END. New tile graphics for mapping can be added by placing 64x64 graphics between G\_START and G\_END lumps. | If after editing the .WAD and saving the changes, all the additions appear in the game as the player added/edited in .WAD, then this test is passed. |
| **Level Progression**. At the end of each level, there are blue tiles that will send the player to the next level | This test passes if the player is sent to the succeeding level upon reaching the blue progression tiles at the end of the current level |