**SOFTWARE DEVELOPMENT PLAN**

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

**REVISION HISTORY**

|  |  |  |  |
| --- | --- | --- | --- |
| Revision # | Author | Revision Date | Comments |
| 1.0 | Isaac Colon,  Meshwa Patel,  Yifan Zhang,  Kwadwo Gyasi-Danquah  Parth Patel | September 17, 2021 | initiated |
| 2.0 | Parth Patel, Isaac Colon | November 21, 2021 | Updated software overview, features and requirements  Revised Activities, development environment |
|  |  |  |  |
|  |  |  |  |

**Table of Contents**

[System Overview 4](#_Toc88425527)

[Document Overview 6](#_Toc88425528)

[General Requirements 6](#_Toc88425529)

[Features and Requirements 6](#_Toc88425530)

[Activities 7](#_Toc88425531)

[Requirements Gathering 7](#_Toc88425532)

[Top-Level Design 7](#_Toc88425533)

[Detailed Design 8](#_Toc88425534)

[Testing 9](#_Toc88425535)

[Tasks 9](#_Toc88425536)

[Schedule 12](#_Toc88425537)

[Milestones 13](#_Toc88425538)

[Development Environment 13](#_Toc88425539)

[Version Control 14](#_Toc88425540)

## 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 when playing in multiplayer mode. 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.

## Document Overview

This document describes the design and features of the game, tasks to be performed, the schedule and milestones to follow, and development requirements to complete the game. It includes the following sections:

* System Overview
* General Requirements
* Features and Requirements
* Activities
* Tasks
* Schedule
* Milestone
* Development Environment
* Version Control

## 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

## Features and Requirements

* Space Crucibles will have a main menu that will be presented when players load into the game
  + The main menu will have five options: Single player, Co-op, Level Editor, Settings, Exit
* The player will be able to enter their in-game username in the settings menu
* Each predefined level in the game will have five difficulties, Very Easy, Easy, Medium, Hard, and Nightmare
  + Each difficulty level will get progressively harder by spawning more monsters, monsters will respawn in harder levels, monsters will have increased health, and be able to run faster.
* The Single player option will show a level difficulty window that will allow the user to select the difficulty of the level
  + Once a difficulty is selected the player will spawn in-game and will be ready to play the game
* The Co-op menu will have three options: Create Lobby, Join Lobby, and Back
  + In the Create Lobby option the player will be able to share a four-digit letter/numeric lobby code that routes to the IP address and port number of the lobby
  + In the Join Lobby option, the player will be able to input the lobby code to join their friend’s lobby
  + The Back option will bring them back to the main lobby
* In the lobby, the lobby host will be able to change the difficulty level of the game
* The level editor will allow a user to create a new level from scratch using game assets
* If the player is playing a game in co-op mode leaves the game a bot/AI player will replace the player and help the teammate finish the level.
* The ESC key menu will have two options: Restart Level, and Return To Title Screen
* In the settings menu the player will be able to change the volume of in-game sounds and music, change username, and add addons
* The lobbies have no limit to how many players can join so the user can invite unlimited number of players to the lobby
* The game will allow new players to join in middle of co-op game
* In both single-player and multi-player modes the players will have a minimap that has the layout for the whole map.
* The game loop will run at 55 ticks per second to improve input precision
* The players will be able to create their own levels and add custom entities to the game
* The players will be able to play their custom levels with each other in co-op mode.
* Different types of monsters with different abilities will be introduced to the game
* There will be an in-game chat that players will be able to communicate through by typing.

## Activities

### Requirements Gathering

First, an IDE that can compile Java programs is needed, followed by a basic understanding of game design in LibGDX. We decided to use IntelliJ as the IDE to create the project. We will need to download LibGDX and import it as a library into the libraries folder of our project’s package. This will give us most of the necessary classes and functions needed for developing a top-down 2D game, though we will need to implement some additional code that caters to our own game’s specific needs. This includes on-fly-code we directly write ourselves or scripts from online resources for more complex features. LibGDX. is compatible with Windows, Linux, and Mac OSes, so we will be able to create a game that’s supported in those OSes. We will also need to import Kryonet, a java networking library, and it will be used to create the communication needed between the server and clients. Before testing begins, however, we will need visual assets to see if the program we have compiled has proper functionality. Similar to the code itself, there are several ways to accomplish this, from simple sprites and tile sets we create ourselves or those from open sources online.

### Top-Level Design

Space Crucibles is a multiplayer game that’s designed to attract users that are interested in playing an action/puzzle game. Once the game is loaded, the user will be presented with the main menu. The main menu will allow the user to choose whether they want to play multiplayer or a single-player mode; the user will also be able to change settings such as key binds and control the volume of the game. In the settings menu, the user will be able to enter a name that will be displayed to other users in-game.

If the user chooses to play the multiplayer mode (co-op), the user will be presented with two options: create lobby and join lobby. The create lobby option will allow the user to create a lobby and give the user (now the lobby host) a generated lobby code, which can be given to friends. The join lobby option will allow other clients to enter this code to join the host’s lobby. The lobby host will have access to kick the players in the lobby. The lobby host will be able to select the level they want to play, choose the difficulty, and also have the option to start the level. Once all the players have spawned in the level, they will be able to press the ESC key to access a menu that will allow them to exit to main menu. If a new player wants to join an ongoing game they will be able to enter the lobby code in join lobby option and join directly.

If the user chooses to play the single-player mode, then the user will be directed to the level difficulty, menu. Once the player chooses the difficulty he will be spawned inside the level. The player will be able to press the ESC key in-game to access a menu that will allow them to restart the level and quit the game.

The gameplay will consist of players trying to clear the level in the game. Each level will have five difficulty levels, each getting progressively harder by including more traps and more/harder monsters. The player’s goal is to clear the level by exploring the map and finding the exit.

The level editor option in the main menu will allow the user to access the level editor and be able to create or customize their own levels. The saved levels will be stored in a .WAD file that the user will be able to share with other players to allow them to play the game. The players will be able to load their own custom levels by going into the settings menu and selecting addons option. From the addons menu, they will select the level file they want to load. Once added, the player will be able to play that level in either single-player mode or multiplayer mode.

### Detailed Design

The design of Space Crucibles follows client-to-server connections to run different facets of the game. On the server-side, all non-player-controlled game events are handled. The server controls the artificial intelligence of enemies, traps, and non-player characters. However, this is not the case for single-player mode, the whole game is run on the client side and there is no need for an internet connection to play the game. Additionally, the server loads different level data from WAD files, including physical enemy, trap, and map geometry placement. On the client-side, hotkey input, graphic rendering, and music playback are handled. The player only controls their character’s actions via keyboard and mouse input, sending that data to the server so it can update the player’s position accordingly for physics calculations.

The simple .WAD archive format is the file type used to store all game data. It stores level texture data (i.e., what graphics will represent the walls and floors in the level), monster and player graphics, other graphic effects, sound effects, and level music in MIDI format. A .WAD file also stores level data. The level data is stored in a text format that records the locations of all walls, floor tiles, monsters, switches, traps, props, and player spawn points in the level. The level editor allows players to edit this data in an easy-to-use GUI, exporting the level in a raw text file. Many of these files can be stored in one .WAD. In addition, new game objects (known as Entities) can be created in a .WAD file in an ENTITIES script. This allows new monsters, projectiles, and other game objects to be created by a .WAD author without writing new Java code. Instead, the new Entities are loaded at runtime when the .WAD is loaded as an add-on file. Then, in the level editor, these new Entities can be added to user-created levels.

### Testing

The testing process will be divided between client, server, and editor tests. For server testing, the first step is making sure the game accepts all connections and proceeds with the game at all with two to four players. If the game can launch, further server-side tests must be performed to analyze game performance and to ensure that all players and the server are perfectly synchronized. The server should also be run with the game on all possible difficulties and other game settings to make sure that the server is reading any map data specific to these settings (for example, the harder difficulties will spawn stronger and numerous monsters). On the client-side, the heads-up display should be tested to check if it matches the player’s corresponding data on the server-side. Additionally, one of the most important client tests is to ensure that the server receives all player inputs, so that player movement and combat are smooth. Finally, a system for testing whether map files are in fact valid, playable levels should be implemented to make sure that changes made in the level editor are reflected in the output file.

## Tasks

**Planning/ Elaboration Phase (PP)**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **#** | **Task** | **Predecessor Tasks** | **Estimated Effort** | **Finish Data** | **Assigned Individuals** | **Successor Tasks** |
| 1 | Brainstorming | None | Everyone | Ideas | Everyone | Any |
| 2 | Research how to use libGDX | None | Everyone x 2 weeks | Research | Everyone | Any |
| 3 | Mock-up main and in-game menus, lobby, and editor interfaces | None | 2 person x 1 week | Mockups | Isaac | IP9 |
| 4 | Research client-server networking | None | 2 person x 1 week | Research | Parth  Yifan | PP7 |
| 5 | Game design planning | None | 2 person x 2 weeks | Mockups | Everyone | PP6 |
| 6 | Game level format specs | PP5 | 1 person x 1 week |  | Isaac | IP7 |
| 7 | Research optimizing the server to improve performance | PP4 | 2 person x 1 week | Research | Parth  Yifan | Improve game experience |

**Implementation Phase (IP)**

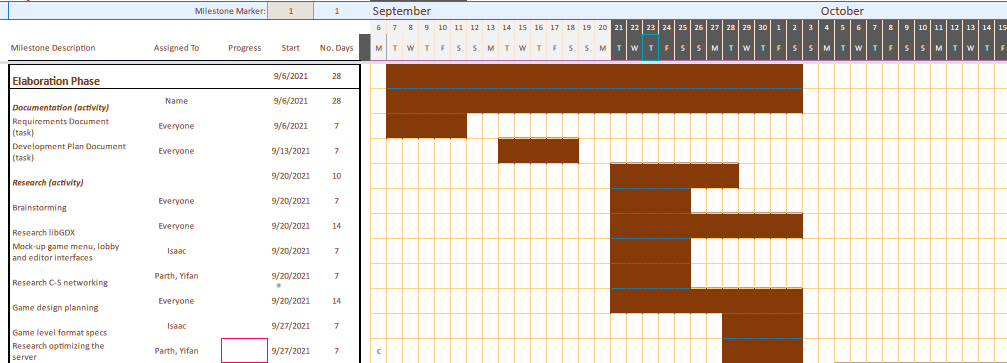
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **#** | **Task** | **Predecessor Tasks** | **Estimated Effort** | **Finish Data** | **Assigned Individuals** | **Successor Tasks** |
| 1 | Implement basic client-server connection | PP4 | 2 person x 2  weeks | Client-server communication | Parth  Yifan | IP2 |
| 2 | Load and run a level with input handling, and collisions | PP2, PP5  IP1 | 2 person  x 1 weeks | Functioning game backend | Isaac  Meshwa | IP3 |
| 3 | Drawing level and player graphics on the screen | PP2, PP5  IP2 | 1 person x 1  weeks | Functioning game frontend / simple gameplay demonstration | Kwadwo | IP4 |
| 4 | Implement traps, and monsters to the game with their physics | PP6 | 1 person  x 1 week | AI-controlled monsters and game traps | Isaac | TP6 |
| 5 | Implement game pick-ups such as weapons, ammo, health, and keys | PP6 | 1 person x 1 week | Player inventory and wieldable weapons | Isaac | TP6 |
| 6 | Implement the ability to clear the level, and loading the next | PP6 | 1 person x 1 week | Complete levels | Isaac | TP6 |
| 7 | Implement a HUD and a game chat | IP1, PP2 | 2 person x 1 week | in-game chat and HUD | Kwadwo  Parth | TP6, TP1 |
| 8 | Create a level editor | PP6 | 1 person x 2 weeks | Level editor | Isaac | TP5 |
| 9 | Implement the lobby feature | IP1 | 2 person x 2 weeks | Lobby to host games | Meshwa  Parth | IP9 |
| 10 | Implement the main menu and escape menu | PP3 | 1 person x 1 week | Complete UI | Meshwa | TP2 |

**Testing Phase (TP)**

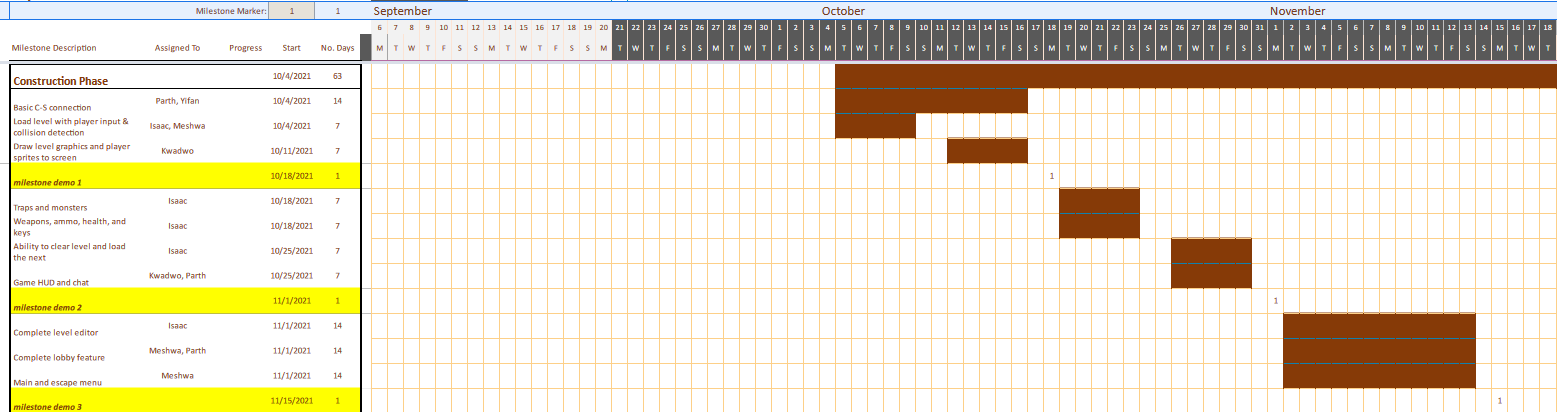
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **#** | **Task** | **Predecessor Tasks** | **Estimated Effort** | **Finish Data** | **Assigned Individuals** | **Successor Tasks** |
| 1 | Back-end connection test | IP1, IP8 | 1 person x 1 week | Back-end system | Parth | TP2, TP3 |
| 2 | Front-end design test | IP3, TP1 | 1 person x 1 week | Game functionality | Meshwa | TP4 |
| 3 | Game synchronization test | IP2, IP5, TP1 | 1 person x 1 week | Game usability | Parth | TP4 |
| 4 | Game performance test (FPS) | TP2, TP3 | 1 person x 1 week | Game usability | Yifan | None |
| 5 | Design user-made test levels | IP8 | 1 person x 1 week | New level | Isaac | None |
| 6 | Game logic test | IP4, IP5, IP6, IP7 | 2 person x 1 week | Game functionality | Isaac  Kwadwo | None |

## Schedule

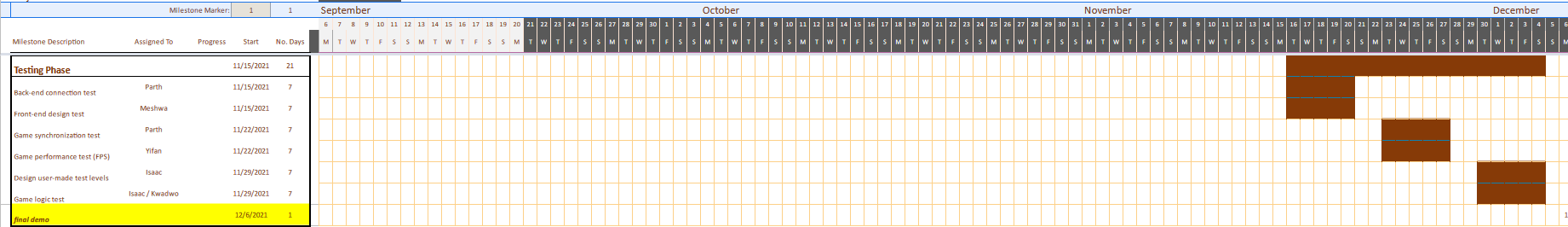
**Pre-Implementation Phase**

****

**Implementation Phase**

****

**Testing Phase**



## Milestones

|  |  |
| --- | --- |
| **Deliverables** | **Date** |
| Players will be able to connect to the server and move around in the level, basic collision detection i.e. bumping into players and walls | 10/18/2021 |
| The player will be able to use weapons, the levels will have traps and monsters that the player will be able to fight.  The player will be able to view their health, how much ammo they have, etc. The player will also be able to chat in-game. | 11/1/2021 |
| Basic level editor implementation and user WAD data loading. Users enter a lobby before starting the game. | 11/15/2021 |
| Space Crucible will be fully developed with all its features | 12/06/2021 |

## Development Environment

**Required Hardware**

We will need computers that have Windows, Linux, and macOS in them because we will be testing on those OSes.

**Required Software**

LibGDX. setup tool

Kryonet library

Doomstruct library

**Selected IDE**

IntelliJ IDEA, Version: 2021.2.2 or higher

**Compilers**

Javac 1.8

**Test Tools**

Junit

**Build Tools**

Gradle

## Version Control

Version Control Tool: Git Repository

We are going to use Git Repository for the version control. It is a distributed version control system which means that each developer has the full version history. There are two components of GitHub, the working directory and the repository. The working directory consists of the branches that we create and on the ones that we are working on. The repository consists of the staging area and commits history. We can use the git add command to stage files in the staffing area. From the staged files, we can choose what file and the changes we wish to commit. The ones that we want to commit can be done using the git commit command. Once the changes are committed, depending on the permissions, the files must be reviewed by peers who are a part of the project. All the remaining members can then pull the new commits, and this is when they have the full new version of the project, which they can now start working on.