Team 20 Project Design Document

Controlled Chaos

Team Members:

Karina Abraham, Bolun Zhang, Cameron Hofbauer,

Javad Baghirov, Zayden Newquist, Jack Wagner

Index

Purpose

The goal of this project is to make an entertaining game that balances a curated experience and randomly generated content. Controlled Chaos will feature the core tenets of a roguelike game, including randomly generated levels, enemy encounters, and item discovery. Our game will be different from other games by incorporating themes of randomness in the level design/artwork and by having dynamic item functions that are discovered and documented in-game by the players themselves.

Functional Requirements

Functional requirements for our project may be found in our project backlog:

Backlog

Non-Functional Requirements

As a developer, I would like to…

Game Engine

- Drag and drop game objects into a certain scene like the main character and enemy non-playable characters.
- Apply animations to game objects using the game engine.
- Build levels by dragging and dropping tiles.
- Add sprites to the game objects using the game engine

Gameplay

- Make the character accelerate when the player starts moving and decelerate when the player stops.
- Have specific enemy types in only certain room types.
- Have the option to enable and disable vertical sync.
- See animations which match up with a specific event, i.e. barrels destroyed only when the user's weapon hits it.
- Hear sounds which match up with a specific event, i.e. attack sounds in the middle of the attack animation.
- Get the latest version of the game from a source control.
- Make the tutorial easy to understand.

- Have my progress saved in a JSON file from Java objects.
- Have a cooldown between sequential attacks.
- Have at least 30 fps everywhere in the game except for title screens and the menu screen.

Design Outline

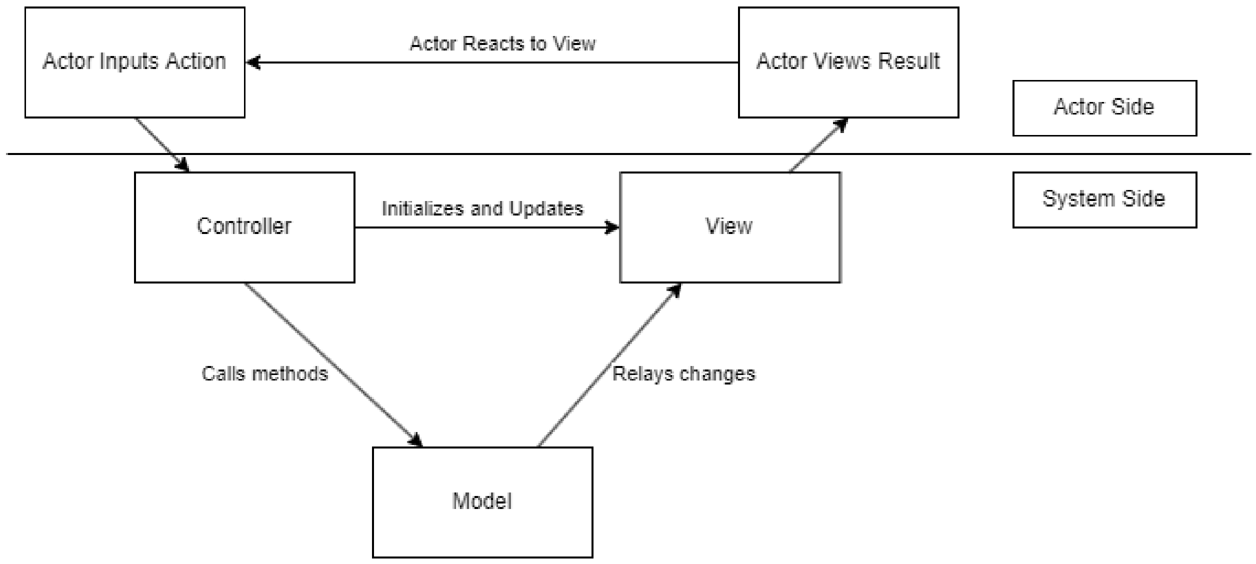
High Level Overview

The deliverable for this project will be an executable for a rogue-like game called Controlled Chaos. The user will not need access to the internet or other remote services to play the game. This application will be constructed using the Model-View-Controller architectural pattern.

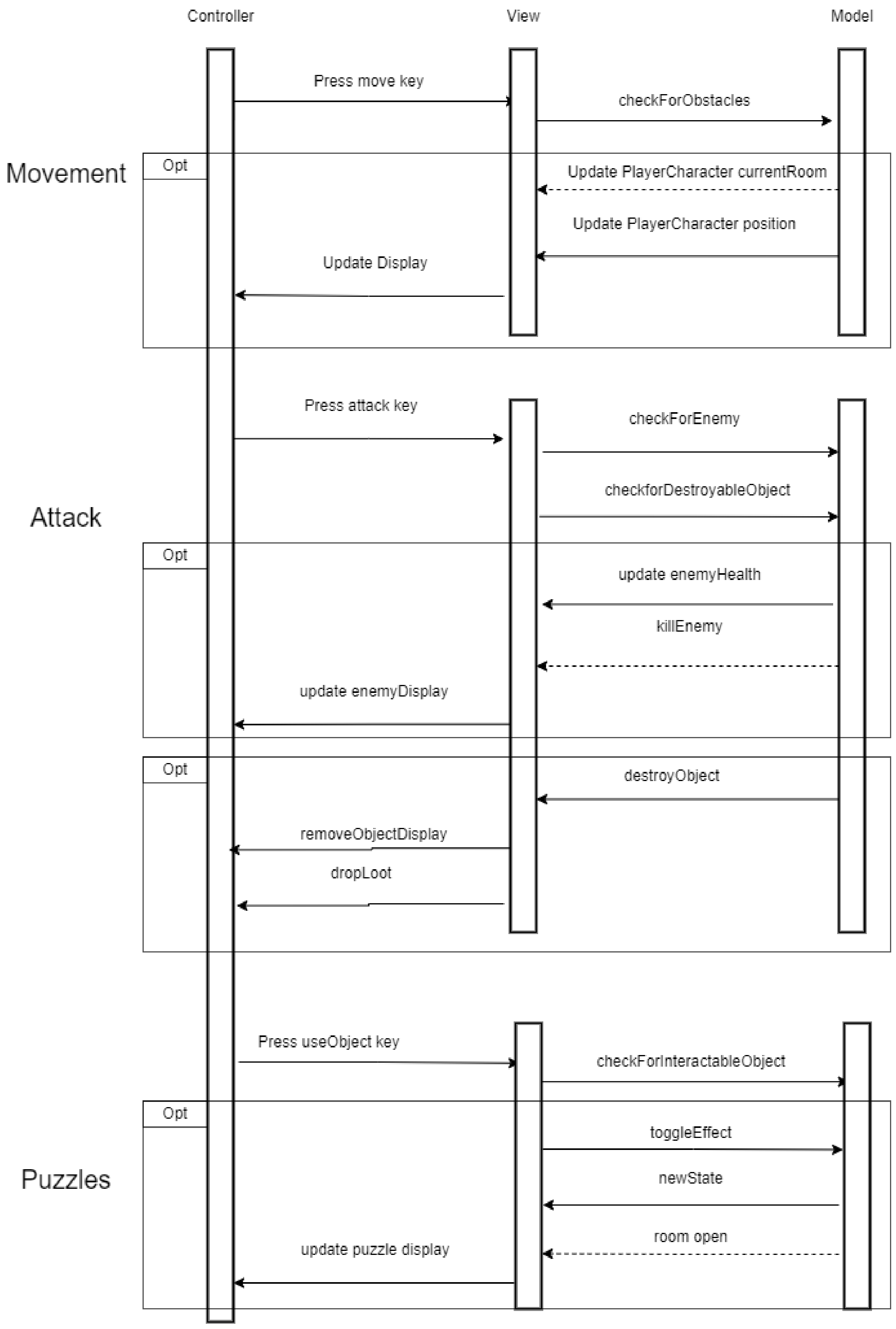
The Model component is responsible for most of the logic behind the program in question, as the objects of its classes are manipulated as necessary. For this application, it will consist of all objects that contain logic for major portions of the game engine. This will include the Character, Room, and Item classes. Its classes will be initialized at startup of the game, and the classes' methods will be called by the Controller when the user interacts with the game, i.e. acquiring items or entering rooms. This component will likely require the most amount of work as it forms the basis of the game itself.

The View component is responsible for the logic behind the user interface. For this application, it will contain objects that can be used to render displays of characters, items, rooms, and other important components of the game. Like the Model component, its classes will be initialized at startup of the game, but it can be called by both the Model and the Controller when an object's visual component needs to be updated.

The Controller component is responsible for handling responses to input from the user. For this application, it will likely consist of a single object which runs the entire program. It will be initialized upon startup and, given user input such as pressed keys, it will redirect execution to either the Model or View components. In both options, the user interface will be updated through the objects in the View component. It is possible that this portion of the game engine can be constructed using two threads - one that listens for user input, and one that redirects execution to the other two components.


Actor Inputs Action
Actor Reacts to View
Actor Views Result
Actor Side
System Side
Controller
Initializes and Updates
View
Calls methods
Relays changes
Model

Sequence of Events Overview



Movement

The Player Character is controlled by the keyboard (WASD keys by default). The view component must ask the model if there are any objects in the way of movement. If so, the character will not move. If the character is able to move, they may move into another room, in which case their current room will need to be updated. Whether or not they enter a new room, the character must move to a new position relative to the room, and the display must update with the new position.

Attack

There will be a specific key for the attack action, which the controller will be checking for. An attack will have some effect only if there is either an Enemy or a Destroyable Object nearby to be hit. If an enemy is within range, it must take damage; if the damage done by the player is more than it has left, it will die. The display needs to be updated, either with the Enemy's new health bar or with it disappearing. If a Destroyable Object is hit, the object will be destroyed and removed, and the loot inside will be dropped for the player to pick up.

Use Object (Puzzles)

When the controller detects that the user pressed the push object key, the model must check to see if there is a usable object nearby. If so, the object should toggle the effect over which it has control (for example, set a door state to open). The new state of the object must be communicated to the display, and if the proper conditions are met, the next room becomes available.

Design Issues

Functional Issues

Functional Issue #1: What type of rogue-like game should we develop?

- Option 1: Platformer
- Option 2: Top-down
- Choice: Top-down
- Discussion: After coming to the initial decision to develop a rogue-like game, we decided to go with a top-down design approach for several reasons. First, we examined other existing titles, looking at platformers and top-down games in the genre. We looked at the rogue-like platformer Spelunky, noting that the skill of the game was largely based around fast-paced jumping and combat, and that it was one of few platformers in the genre. The variety of other titles which followed a top-down approach focused on allowing the player to curate their items and more strategically navigate combat. We came to the conclusion that top-down design allows for much more depth of gameplay, and therefore a more nuanced project to develop, which is why we chose this genre of game.

Functional Issue #2: How does the user navigate the inventory to write object functions?

- Option 1: Click on items with the mouse
- Option 2: Navigate items with the arrow keys
- Choice: Navigate items with the arrow keys
- Discussion: We chose to have the user navigate their inventory using the arrow keys so that the game is more streamlined. If the users were required to use the mouse, they would need to interrupt the flow of the game to move their hands to the mouse. Though this is not a big issue, it keeps the game accessible through just the keyboard.

Functional Issue #3: How does the user interact with the map, given that the outlines of the rooms are included?

- Option 1: Use the keyboard to draw on the map
- Option 2: Use a mouse or trackpad to draw on the map
- Option 3: Have users type notes on the map

- Choice: Type notes on the map
- Discussion: The user should be able to use the map to document things such as items forgotten in certain rooms, enemies in each room, etc. Options 1 and 2 may invite an unnecessary amount of difficulty; drawing with a mouse, trackpad, or keyboard may prove tiresome to the user, enough to make him/her not want to use the map feature. Option 3 will allow the user to document issues without requiring too much effort from the user.

Functional Issue #4: Should we have a leveling up system, where the player gains strength or health when they gather enough experience from enemies?

- Option 1: Yes
- Option 2: No
- Choice: No
- Discussion: We chose to omit a leveling up system because the same functionality can be achieved using items. Games with leveling up systems are in place to encourage players to defeat difficult enemies as well as to give a sense of progression. In our game, the player cannot progress to a new room unless they have defeated all the enemies, so a level requirement is not necessary. Other level-up progressions, such as added strength or health, will be encountered through items as well. Also, by having enemies drop items when defeated, players are rewarded for exploring the entire level.

Non-Functional Issues

Non-functional Issue #1: What programming environment should we use to develop our game?

- Option 1: Unity
- Option 2: Java
- Option 3: C/C++
- Choice: Java for our programming environment
- Discussion: Some of our group members have little to no experience with development using Unity. However, everyone in our group has taken classes in Object-Oriented Programming and C-Programming. Further discussion makes it clear that Java would be the more comfortable choice based on our experience. Another reason is that Java has

more abundant APIs that would be beneficial for game development. Therefore, we unanimously decided to choose Java as our programming language to build our game.

<u>Non-Functional Issue #2</u>: What protocol should we implement to save specific game states?

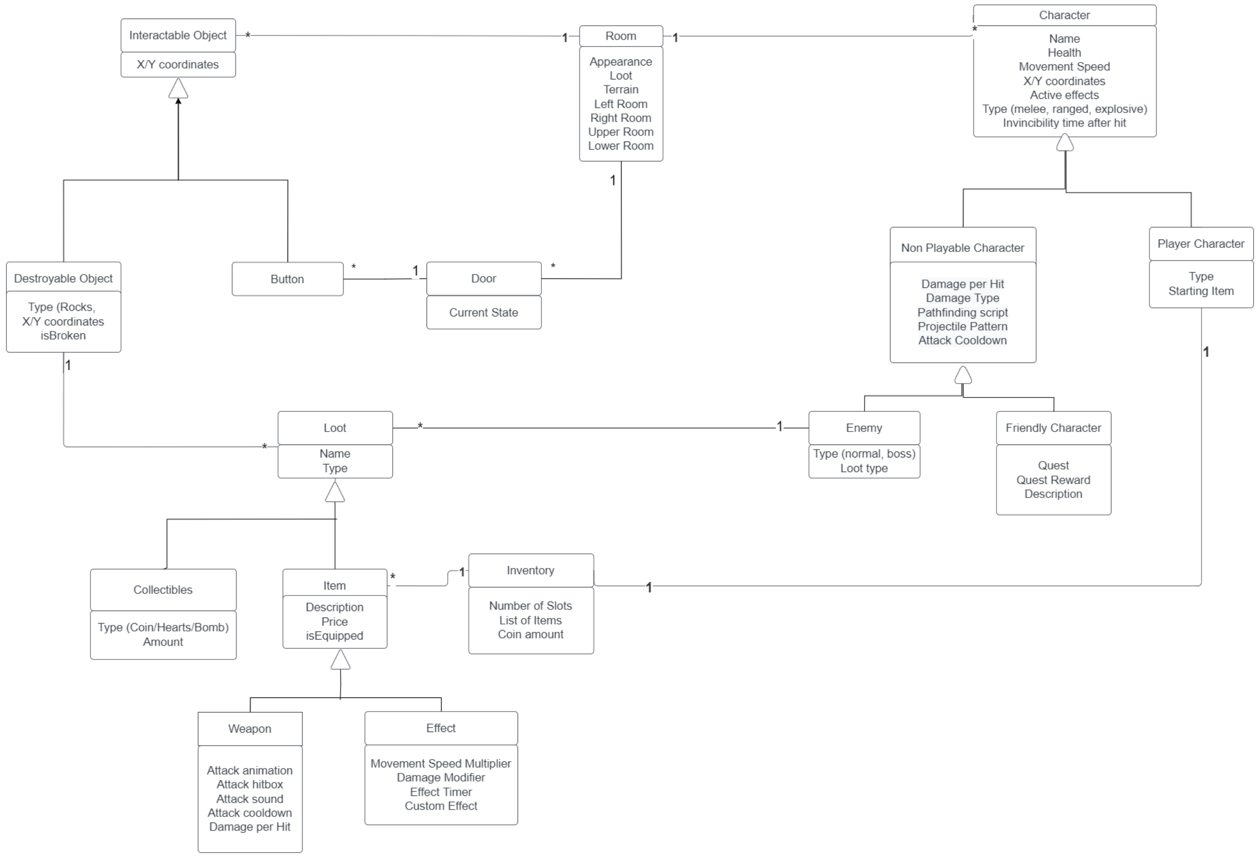
- Option 1: JSON API
- Option 2: Develop a game-specific grammar
- Choice: JSON API
- Discussion: Developing a game-specific grammar for backend tasks, like saving game states and saving player scores, is essential for creating a robust game experience. On the other hand, there are plenty of JSON APIs for us to use to save game states effectively. As software developers, we would rather use an API for which the file reading/writing protocol is already in use and regularly maintained. Furthermore, due to time constraints, we would rather focus our efforts more toward the game engine rather than creating a new file saving function. Therefore, we decided to use a JSON API to save specific game states.

<u>Non-Functional Issue #3</u>: How should we generate the layout of the rooms in terms of how they connect to each other?

- Option 1: Link rooms together in a specified layout and have a displayed map
- Option 2: Have only one exit and entrance per room with no layout or map
- Choice: Link rooms together
- Discussion: We decided to have rooms linked together since this adds depth to the gameplay, allowing users to navigate previously explored rooms. This adds the ability for us to implement puzzles which span multiple rooms, complex secrets, and backtracking for health that was left behind. Developing level generation also adds complexity to the project, since it requires an algorithm to build a layout out of the rooms we develop.

Design Details

Core Gameplay Loop Design



Character Class

The Character class contains basic information about a character; it is the parent class of the Player Character and Non Player Characters. Non Player Character also has subclasses Enemy and Friendly Character. The Player Character will be controlled with user input from the keyboard. Enemies will be controlled by a script which involves moving towards and attacking the player character. Friendly Characters will not attack the Player Character, but will instead offer quests which can be completed to obtain loot.

Loot Class

The Loot class is responsible for holding information for various items within the game. It is the parent class of the Collectibles and Item classes. The Item class is specifically meant for objects that require descriptions and/or prices. It is also the parent class of the Weapon and Effect classes. Each of these classes have their own fields specific to each type. The inventory class contains a list of items that the player character in question currently has in his/her possession. (Each player character has its own inventory object.) The other type of object that is a child of the Loot class is the Collectibles class, which is to be used for health, coins, or bombs.

Interactable Object Class

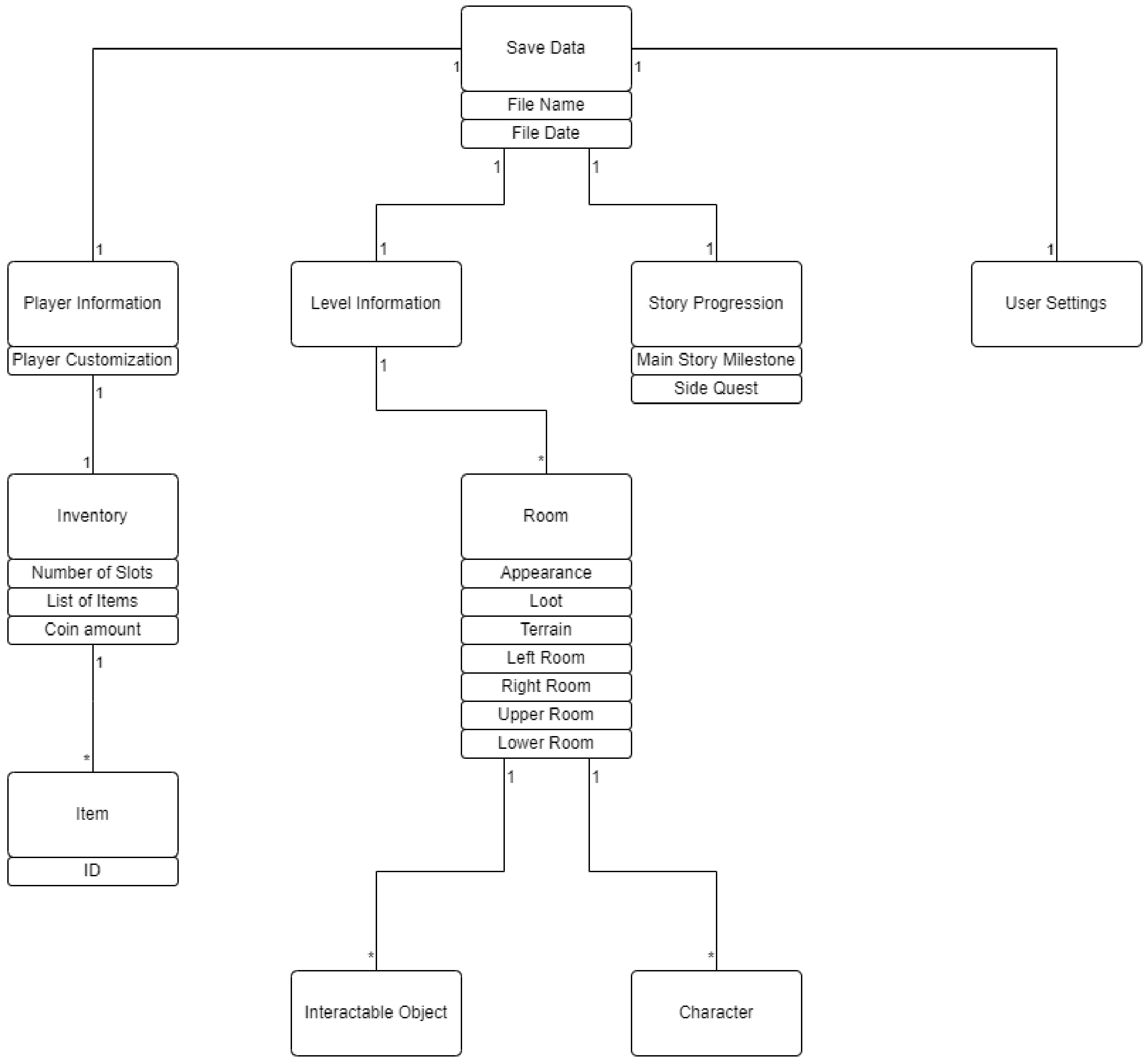
The Interactable Object class is the parent class of Destroyable Object class and Button class. Buttons can be pressed to open doors; they will be used to implement puzzles for the user to solve. The Destroyable Object class are objects the player may come across which have loot inside and can be destroyed to obtain this loot.

Room and Door Classes

The Room and Door classes have no child classes. The layout will be generated using many Rooms, and each Room may be connected to up to four other rooms on the left, right, top, and bottom doorways. Any given room may have several Doors which lead to it. It is also possible that a Room has no Doors and is instead accessed through some other means (such as teleportation) as part of a puzzle or hidden feature. A Door object will contain its current state, either open or closed.

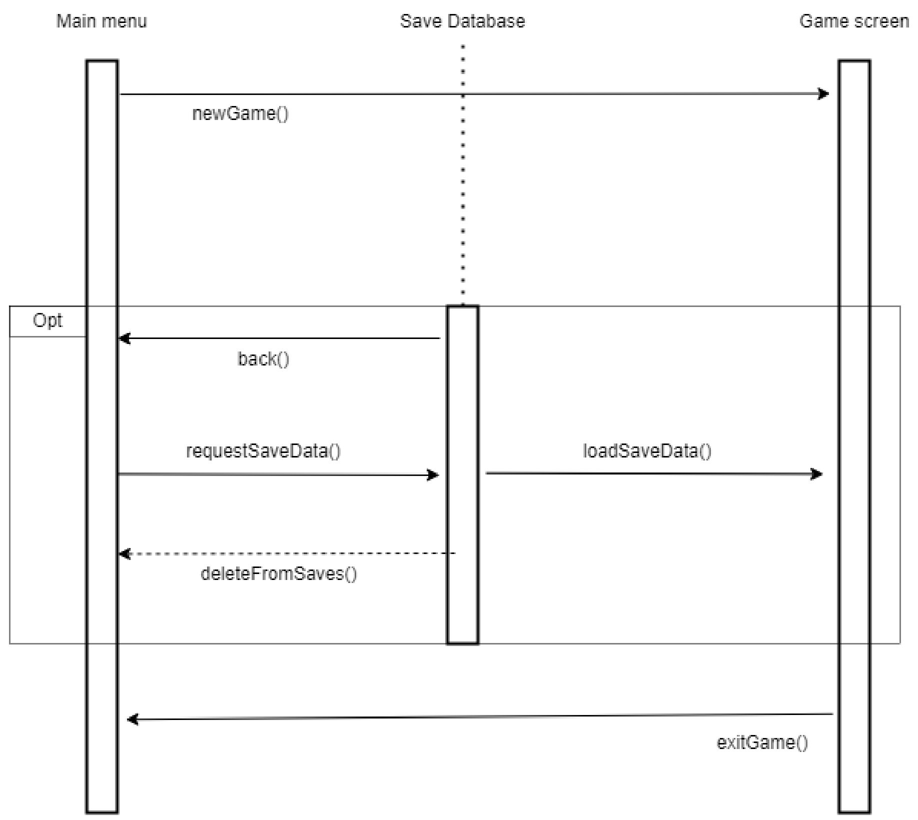
Data Persistence Architecture

The logic that controls saving data from the game will be separate from the core gameplay loop but will be part of the Model component of the overall architecture. The Save Data class contains fields that hold the name of the JSON file which contains the data and the date when the file was created. This object will hold objects that contain data on leveling up of the player, story progression, accessed rooms, player information, items held by the player in the inventory, and settings indicated by the user.

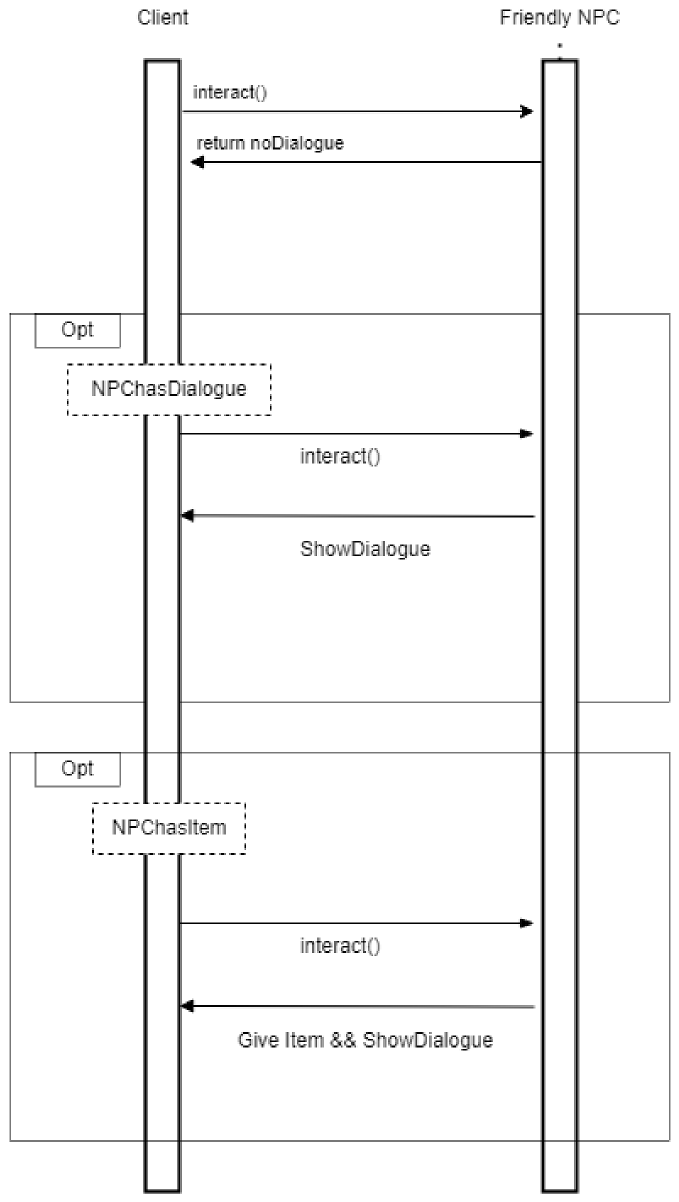


Sequence Diagrams

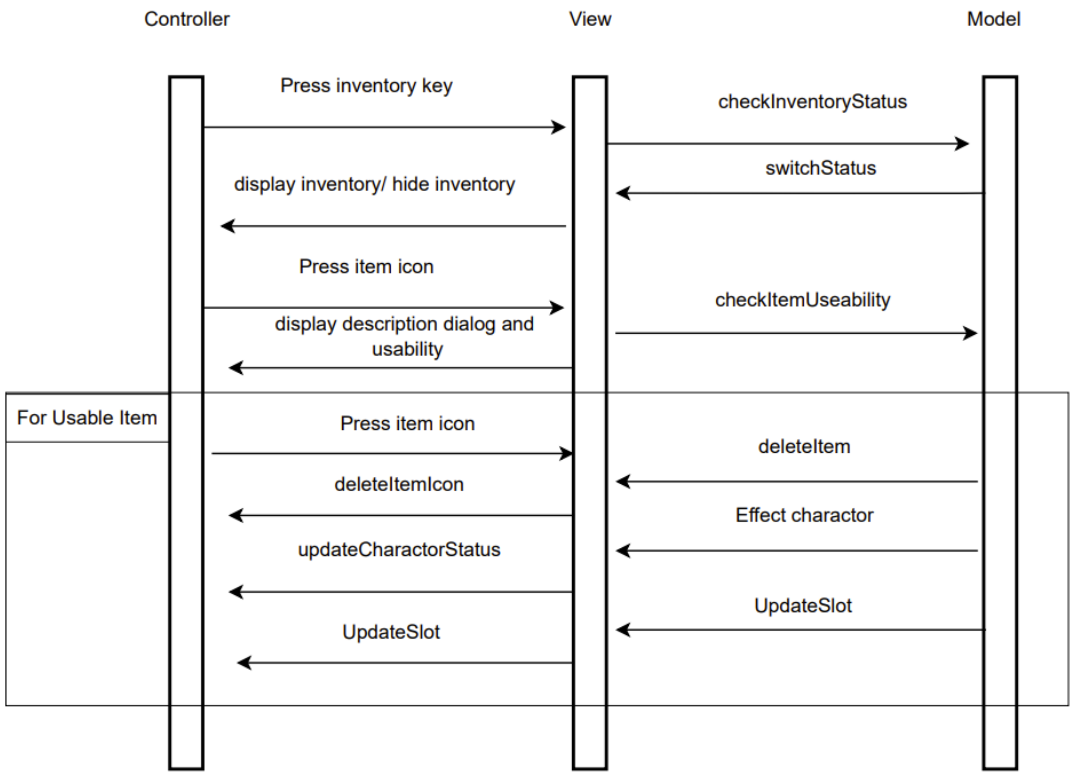
Save Database Diagram



Non-Playable Character Interaction Diagram

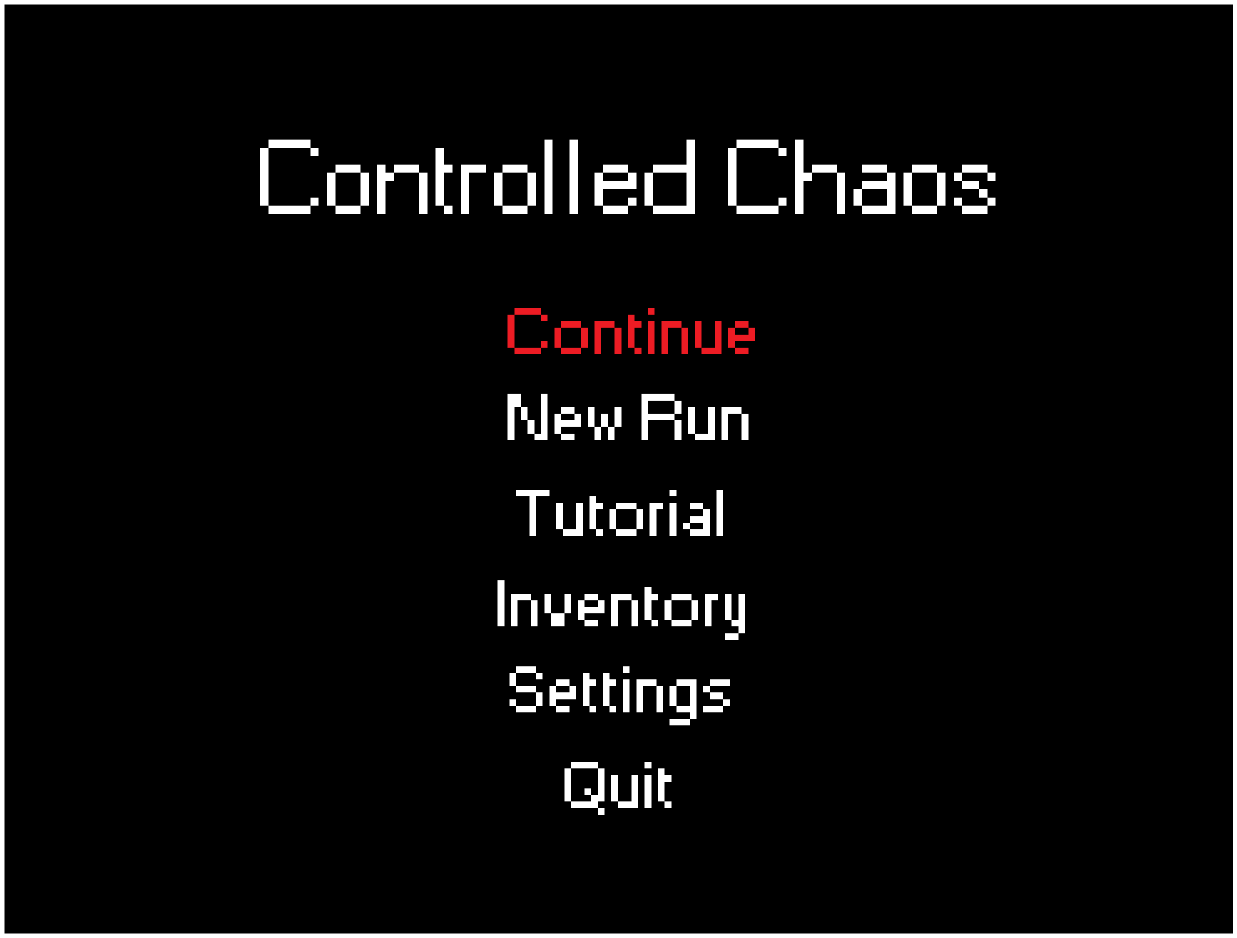


Inventory Management Diagram

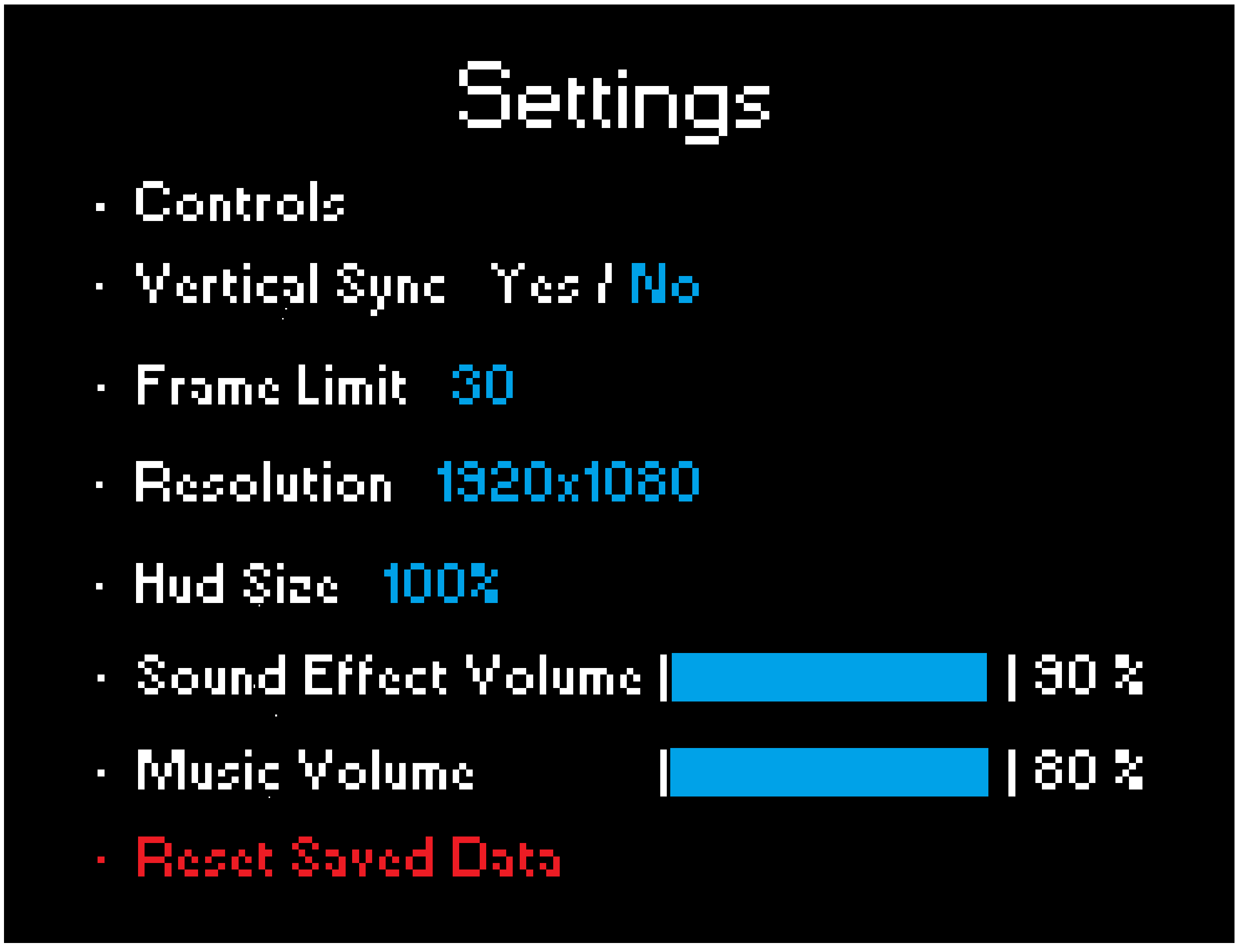


UI Mockup

Title Page:



Settings:



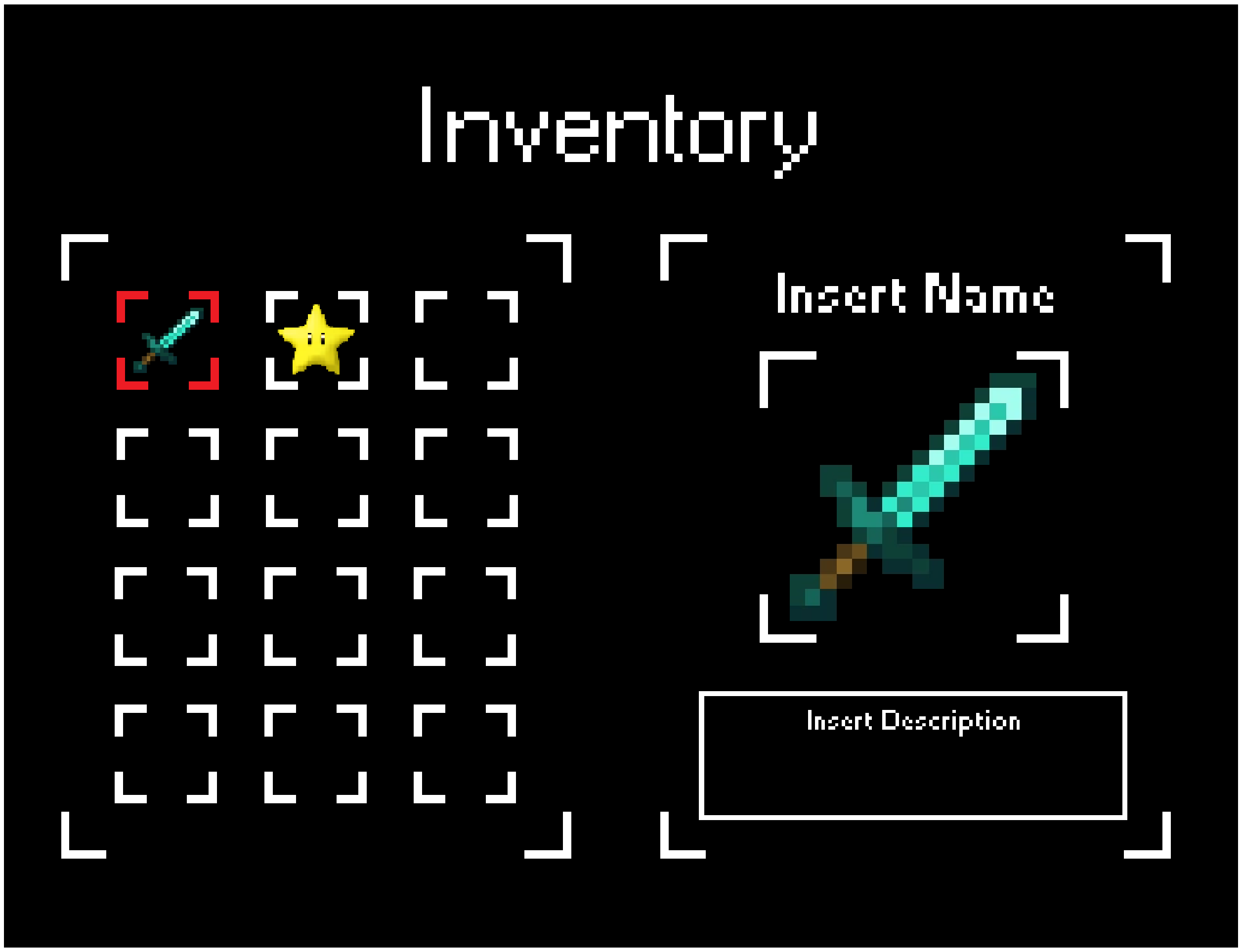
Settings

- Controls
- Vertical Sync Yes / No
- Frame Limit 30
- Resolution 1920x1080
- Hud Size 100%
- Sound Effect Volume | | 90 %
- Music Volume | | 80 %
- Reset Saved Data

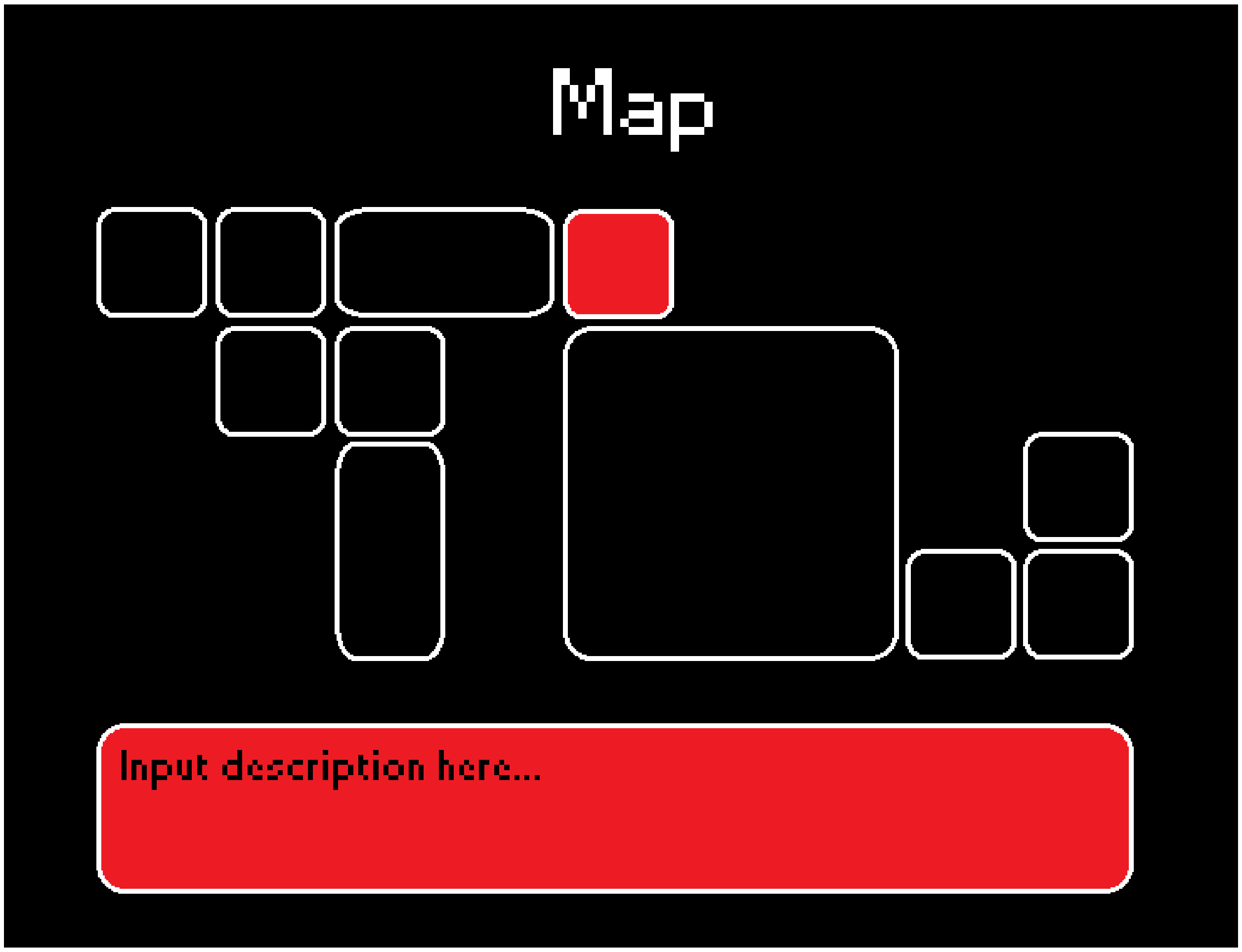
Inventory:

Note that the icons were taken from the Minecraft and Super Mario games. Here are the links to the icons:

- https://minecraft.fandom.com/wiki/Sword
- https://iconarchive.com/show/super-mario-icons-by-ph03nyx/Star-icon.html



Map:



Room:

Note that the icons and characters were taken from various online sources. Here are the links to the web pages:

- https://www.pinterest.com/pin/62698619783142458/?nic_v3=1a4VFUmxZ
- https://opengameart.org/content/alien-2d-sprites
- https://www.vectorstock.com/royalty-free-vector/moon-satellite-icon-vector-13340134
- https://www.flaticon.com/free-icon/pit_4677206