**Project Gon**

**Description:** Gon is a 2D top-down view game that manipulates the player’s screen to change the gameplay. The game starts off with the main character, Gon, in search of the 7 light sources in the world. With each orb obtained, the game view expands to allow the player to see more of the world. On his quest to find all 7 orbs, Gon encounters other players and enemy mobs which he must defeat to continue. The game ends when Gon finds all of the light orbs and uses them to defeat the enemy boss.

**Structural Design:** The structural design of the entire project is listed below in Table 1.

|  |  |
| --- | --- |
| **Data** | **Data Structure** |
| Game map | int[][] |
| Logged in players | TreeSet<Player> |
| Registered players | TreeMap<String, Player> |
| Player inventory | int[] |
| Player specifications | int[] |
| Active enemies | TreeSet<Enemy> |
| Enemy specifications | int[] |

*Table 1*

The game map is a 2D array of integers that holds the data for inanimate objects in the map such as walls, orb, and chests. Different maps are written in a simple .txt file and can be loaded in depending on the current level. A TreeSet contains the current Player objects who are logged in while a TreeMap contains all registered players, linking their username to their Player object. An array of integers for each Player objects represents their inventory, with the array index corresponding to the item ID and the int stored representing the number of said item. A second array of integers describes the Player’s specifications such as HP, ammo, armor, and light orbs. Finally a TreeSet of Enemy objects holds a reference to all of the active Enemy objects that are currently active in the game with each Enemy object having an array of integers that describe the Enemy’s specifications such as HP and ID.

**Object-Oriented Design:** Project Gon utilizes an object-oriented design to ensure that features such as adding players and enemies, changing map designs, and many others are easy to implement and do multiple times. Figure 1 below shows how the 7 classes and 2 interfaces work together to make Project Gon a whole.



*Figure 1*

**Class Descriptions:**

|  |  |
| --- | --- |
| **Name** | **Description** |
| ProjectGon | ProjectGon holds the ‘main’ method and runs the entire game. This class holds instances of all the objects and stores the data for the game to run. |
| Map | The Map class utilizes a 2D array of integers to represent the current map which is being played. Different numbers represents different things on the map such as walls, light orbs, and chests. |
| Character <<interface>> | The Character interface writes the outline for Player and Enemy classes to model. This includes generic methods such as attack, block, and move as well as instance variables for the HP and ID. |
| Player | The Player class implements the Character interface and expands on it by creating instance variables for ammo and energy. Also the Player class has an array of integers which represent its inventory and the items it is holding. |
| Enemy | The Enemy class implements the Character interface, utilizing the basic methods and instance variables to function. |
| Item | The Item class represents a single Item object with an ID. |
| GameState <<interface>> | The GameState interface includes the basic methods necessary for all states such as update. The GameState also has an instance variable to hold the background image or color of the state. |
| MenuState | The MenuState class implements the GameState interface and adds buttons to allow the user to choose what they would like to do at the start of the game. |
| LevelState | The LevelState class implements the GameState interface and holds an instance variable for the Map. The LevelState also keeps track of all the Enemy and Player objects currently active. |

**Workflow:**

1. Write the Character interface and Player and Enemy classes. This should be fairly simple with very little methods and instance variables to write out.
2. Write the Item class which should also be very simple.
3. Write the GameState interface and MenuState and LevelState classes. Designs should be sketched out before hand for how we want each level and the menu to look.
4. Write the map class and multiple sample .txt files to be used as maps.
5. Bring together all the classes in the ProjectGon class which manages all the objects and also contains the images for each Player or Enemy.
6. Create the JUnit testing class to test functions of the program.
7. Write documentation for each class and describe the purpose of each method.