**CS112 Final Task**

**Text-based RPG**

Name: Youssef Zein ID: 231001462 (Team leader)

Name: Youssef Khaled Ali ID: 231001203

Name: Jomana Hussein ID: 231001427

Name: Ahmad Elmoghraby ID: 231000713

Name: Mohamed Salama ID: 231000359

In this project, we aimed to create our own version of a text-based role-playing game. The game immerses the user in an explorable world, allowing the user to interact with the game by prompting them through a set of provided options. The biggest challenge/obstacle faced when starting was understanding how we are going to allow the user to maneuver through the game, and how it’s going to further progress.

The game starts by asking for the user’s details, followed by a main menu in a function called “IdleMenu”, which lists options that guide the player throughout the game. providing them with 6 options: “Embark on a journey”, “Inventory”, “Shop”,Stats”, “Save”, “Quit”.

Firstly, pressing option 1, “Embark on a journey” calls the “LocationIntro” function, which displays a set of locations for the user to pick from, and enemies are assigned based on the location the player chooses to explore, meaning that each area has its designated enemies/mobs. This was done by creating a class named “Enemy”, and making an array of objects with the data type Enemy, each index in this array represents a different enemy (object) which different stats. To generate the enemies, the player’s choice of location is required to indicate the index of the object in the array of enemies that will be copied into a new Enemy object, therefore generating a new enemy. In addition, to ensure that the game keeps progressing with the player, we made sure that public members of the enemy class such as “Xp” and “LVL” were initialized based on the player’s Xp multiplied according to a ratio. Afterwards, the FightMenu function is called, showing the player’s health, level, and name, along with the option to fight, use items, or escape. Picking “fight” checks whether the enemy’s health has reached zero, which increases the player’s XP and cash, it also checks the player's XP and levels up the user accordingly, and lastly if the enemy hasn’t been defeated, the function EnemyReaction is called, allowing the enemy to either heal itself, or deal damage to the player by attacking them, and possibly defeating the player. Moreover, the option “Escape” randomly decides whether the user successfully escapes, hence leading them back to the idle menu, or fails to do so, consequently triggering the enemy’s reaction once again. Choosing to use items runs the use\_consumable function, which checks the validity of the input, whether the item is even available in the player’s inventory. If both cases are true, the item is used to increase the player’s health while making sure it doesn’t exceed the player’s maximum health. Unexpected cases are avoided by checking the validity of the user’s input first, then outputting a simple message prompting the user back to the idle menu.

Secondly, the inventory feature allows the user to check their available items by dividing them into the following categories: Weapons, Armors, Consumables. Besides, the user is also provided with the option to go back to the idle menu. Using a switch statement, the user’s choice calls one of the following functions: inv\_weapons(), inv\_Armor(), inv\_cons(). These functions depend mainly on the classes Weapon, Armor, or Consumable, as an array of objects for each of these classes was created. The functions display a list of items by looping through an array of objects either of the class Weapon, Armor, or Consumable, using a for loop, whilst checking whether the item’s amount in the inventory was greater than zero in the first place. (ex: if (Weapons[i].Weapon\_num>0)). The player’s stats change based on their choice of weapon, armor, and consumable, and they’re displayed to the user on the screen. To ensure that the functions are bug-free, we made sure to check all possible inputs and to display error messages while recalling the function.

The player’s stats can be viewed using the third option in the idle menu, which displays all of the data related to the player's health, attack, defense, experience, and level.

Another feature that can be accessed through the idle menu is the shop, which calls a function called “merchant”, which allows the user to either sell items, buy items, or exit and go back to the idle menu. This is performed by a switch statement which calls either of the functions “sell\_items”, “buy”, or “IdleMenu” to exit. To briefly summaries them, both buy() and sell\_items() list the three different categories of items, expecting an input from the user. The program then proceeds by displaying each of these items by looping through the array of objects created for each of these items' classes. However, to ensure that no unexpected behaviors occur, in the sell\_items() function an if condition is implemented to confirm whether the item exists in the inventory to begin with, which is done by checking if number/amount is greater than 0. Additionally, the number entered by the user is checked once more to ensure that the index points to an object that is available in the inventory. This is done to make sure that the player can't sell items that don't exist in their inventory to the merchant. All menus have an additional exit option which prompts the user back to the idle menu. Lastly, details of each item are printed, along with the amount of cash carried by the player, and the cash gained or item acquired.

The fifth option provided allows the user to save or load their stats to an external file created on the pc, and the final option created was the quitting option which terminates the program.

We needed to test our program multiple times and to enter invalid input to examine the unexpected behaviors that can happen, and as a result we had to change and add multiple if conditions to ensure that even if the user enters text, they would still be able to run the program without encountering infinite loops. Our program mostly relied on recursion and calling different functions into other functions.