Report OOP HW1

Abdullah Tariq, 29123

February 28, 2025

1 Introduction

The game was called "Dungeon Escape" where the user had to escape the dungeon with limited moves, health and stamina. I implemented a total of 9 classes, namely:

- `DLL` - Doubly Linked List
- `Dungeon` - The dungeon class
- `Enemy_Queue` - The queue of enemies
- `Enemy` - The enemy class
- `Inventory` - The inventory class
- `Treasure` - The treasure class
- `Player` - The player class
- `Room` - The room class
- `Room_Stack` - The stack of rooms

I choose dynamic memory allocation for most of the classes such as `DLL`, `Enemy_Queue`, `Room_Stack` and `Inventory`. This was done to allow for the flexibility of the number of elements in the list, queue or stack. The Stack and the Queue did not have any resize function since their were no need of it. The `DLL` and `Inventory` classes had a resize function to allow for the resizing of the list.

2 Implementation of the classes

For the **Linked List**, I used the Room class pointer as a Node in the list. The Linked List was doubly since it would easier to traverse the list in both directions.

The **Enemy_Queue** was implemented linearly and each element in it was a pointer to an enemy class object. The queue had a front and a rear pointer to keep track of the front and the rear of the queue.

The **Room_Stack** was implemented linearly and each element in it was a pointer to a room class object. The stack had a top pointer to keep track of the top of the stack.

The **Inventory** was implemented as a dynamic array of pointers to the treasure class objects. Each time a treasure was addded the array was resized to accomodate the new treasure only when the capacity was reached.

Only the **Treasure** class had a overloaded operator which was the `==` operator since I needed to compare the treasures in the inventory while using or removing them.

The **Player** class had a lot of functions to allow the player to interact with the game. The player could attack, defend, use item, and much more. The player's attack function took an enemy pointer as an argument on which the player would attack.

The **main.cpp** file only was used to containt the main function and the game loop. Even the game loop logic was in the Dungeon class and the main function only called the Dungeon class object to start the game. The main file also instantiated the dungeon class, made all the rooms, enemies and treasures and then started the game.

3 Why my functions are where they are now?

I tried to keep the functions in the classes where they made the most sense. For example, the `attack` function was in the player class since the player was the one attacking. I placed the attack enemies function in the dungeon class since the dungeon class object was the master controller of the game and even contained the room. It was easy to call a function from another class even thought the function was in the Dungeon class and absolutely made no sense to have the Implementation their but it was just calling another class's function from within it.

4 Design Choices

I choose to give the player a single option to attack the enemy which was attack all of them together in a room because for me it made sense since I was using a queue. I gave the user a choice between a weapon and some defense item to use in the game, this was done to make the game more interesting and to give the user a choice to make. Although the user could move freely in the dungeon through the rooms, I gave the user a limited number of moves and since moves and health diminishes with each move, the user had to be careful with the moves he made.

This was done so the user has a choice between attacking the enemies or saving himself/herself from the enemies by moving through the rooms gaining treasures and health. Now the question arises, Everytime the user shifted rooms, their is a weapon or a defense item in the room give? The answer is so the user has to move between rooms which consumes moves.

I split all the code in .hpp and .cpp files to make the code more readable and to make sure that the code was not all in one file. I also made sure that the code was properly

commented and the functions were properly documented so that anyone reading the code could understand what the function was doing.

5 Any Challenges faced

The biggest Challenge I faced was the Implementation of the game itself. Like how and what classes should be made and where the methods shall be placed. Took me quite some tome to figure out the classes.

The other challenge was to handle the dynamic memory allocation and deallocation. I had to make sure that I deallocated the memory properly to avoid memory leaks. Although I did make the constructors and destructors work fine for each class, it was still a challenge to make sure that the memory was deallocated properly and for some time I did face issues of memory allocation and undefined behaviour.

The last challenge for me was to debug when necessary. It was always hard to debug whenever their was some error as their were a lot of classes and functions which took a lot of time to do. I had to place so many cout statements to debug the code and it was a pain to do so.

6 Conclusion

The game was a success and I was able to implement the game as I wanted to. The game was fun to play and I enjoyed making it. I learned a lot about dynamic memory allocation and deallocation and how to use it properly. I also learned how to use classes and objects in a better way and how to use them to make a game.