Project 4 – CMSC 202

UMBC Adventure

1. Overview

In this project, you will:

- Practice creating classes with inheritance
- Working with pointers as they pertain to classes
- Using polymorphism
- Working with a large number of classes

2. Background

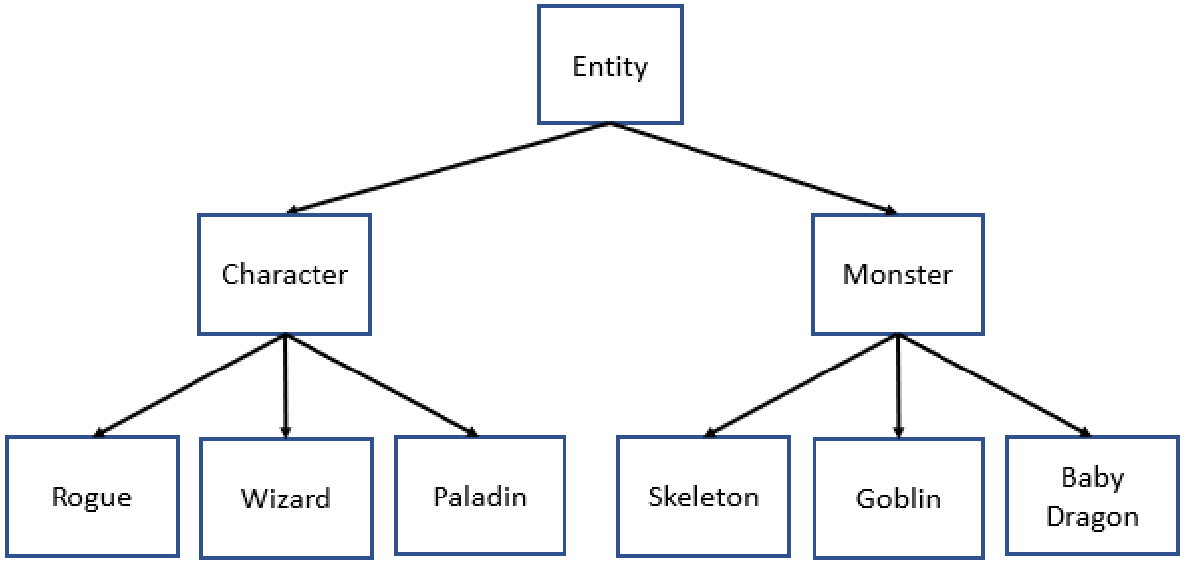
A text game or text-based game is a video game that uses text characters instead of the graphics of modern-based games. The earliest examples of text-based games were introduced in the mid-1970s in games such as Colossal Cave Adventure. They remained popular throughout the 1980s especially as the popularity of Dungeons and Dragons expanded.

Historically, text-based games were popular because they were typically easier to write and required less processing power than their graphics-based counterparts. Even in 2019, people continue playing MUDs (multi-user dungeon) and exploring interactive fiction using text-based interfaces. Many beginning programmers still create these types of games to familiarize themselves with a programming language which is where our project begins.

3. Assignment Description

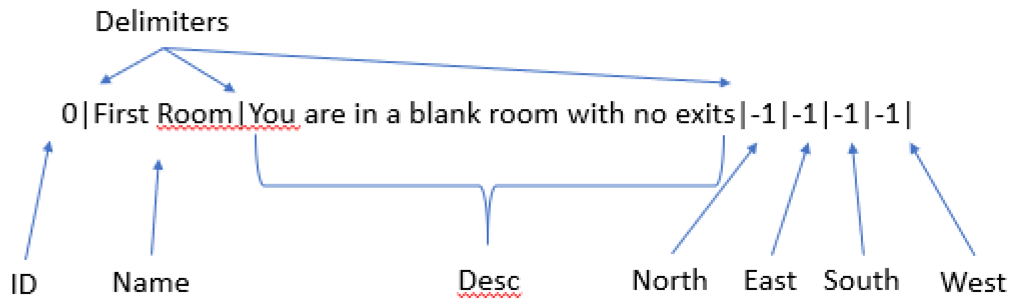
Your assignment is to develop a simple adventure game where a character can adventure through a dungeon and interact with a variety of monsters. Data files for a variety of dungeons will be provided in a standard format.

As this project is focused on inheritance and polymorphism, there is an inheritance chain as defined in the figure below.



In addition to these classes, there is a room class that defines the characteristics for each room and a game class the manages the game itself.

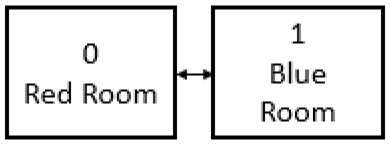
The provided map files are formatted like this:


Delimiters
0|First Room|You are in a blank room with no exits|-1|-1|-1|-1|
ID
Name
Desc
North
East
South
West

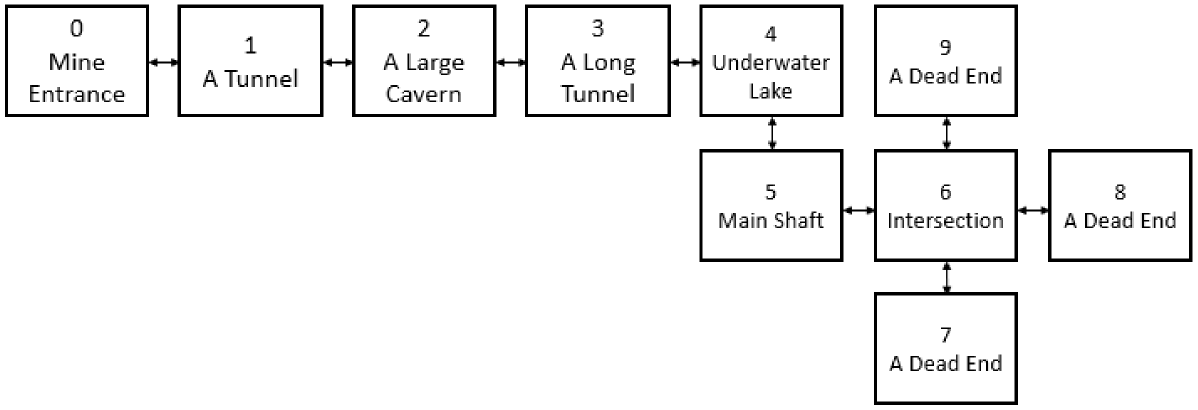
If we look at the first row in the map 1 file, we are saying the following:

1. Column 1 – Room ID (should be an integer)
2. Column 2 – Room Name (string)
3. Column 3 – Room Desc (string)
4. Column 4 – Room ID of room to the north (-1 is no room)
5. Column 5 – Room ID of room to the east (-1 is no room)
6. Column 6 – Room ID of room to the south (-1 is no room)
7. Column 7 – Room ID of room to the west (-1 is no room)

If you are having a tough time visualizing the rooms structures, here is an example of map1_data.txt in an image:



And here is the larger map from map2_data.txt:



4. **Requirements:**

This is a list of the requirements of this application. For this project, it is up to you exactly how you want to implement it. For you to earn all the points, however, you will need to meet all the defined requirements.

- You must follow the coding standard as defined in the CMSC 202 coding standard (found on Blackboard under course materials). This includes comments as required.
- The project must be turned in on time by the deadline listed above.
- The project must be completed in C++. You may not use any libraries or data structures that we have not learned in class. Libraries we have learned include **<iostream>, <fstream>, <iomanip>, <cmath>, <cstdlib>, <vector>, <ctime.h>,** and **<string>**. You should only use **namespace std**.
- You may use any of the conversion functions such as **stoi** or **atoi**.

Using the provided files, **Room.h**, **Character.h**, **Monster.h**, **Paladin.h, Rogue.h, Wizard.h, BabyDragon.h, Goblin.h, Skeleton.h** and **proj4.cpp**, create an adventure game.

- There are ten required classes in this project: Game, Room, Character, Monster, Paladin, Rogue, Wizard, BabyDragon, Goblin, and Skeleton. All functions listed in their class files must be implemented completely (even if you do not use them).

- All entities (and their subclasses) and rooms must be dynamically allocated (and deallocated).
- All user input must be validated. For example, if a menu allows for 1, 2, or 3 to be entered and the user enters a 4, it will re-prompt the user. However, the user is expected to always enter the correct data type. i.e. If the user is asked to enter an integer, they will. If they are asked to enter a character, they will. You do not need to worry about checking for correct data types.
- The name of the map file may change. We have two map files prepared for you to test your program. The name of the files are: **map1_data.txt and map2_data.txt.**

  The map files get increasingly complex (2 rooms then a bunch!)
- You will be using a vector to store the map itself. Each room must be dynamically allocated (and deallocated!).