CS1527 Object-Oriented Programming
2019-2020

Mini-Project: Text-Based Maze Game

Important
This assessment is worth 25% of the overall marks for course CS1527.

Your submission must be submitted via MyAberdeen by **Monday March 9th 2020 at 12:00 noon**.
Late penalties will apply if you miss the deadline.

Remember that as this is an individual assessment, ***work submitted must be your own***.
If you haven't already done so, you should familiarise yourself with the University guidance on plagiarism, available at https://www.abdn.ac.uk/sls/online-resources/avoiding-plagiarism/

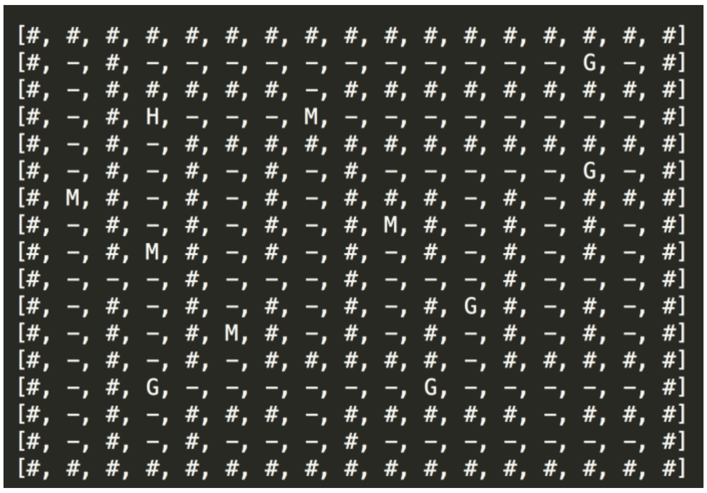
***Note**: If you do not submit this assignment, you will be given a C6.*

Read through this entire document before starting to program.

Introduction:
In this assignment you will implement a simple 2D maze game. This is a text-based adventure game, and the interface is presented on the terminal with printed characters (ASCII code). No fancy GUI is required.

In the game, the hero will explore the maze, fight with monsters, and interact with goblins.

The game will first randomly generate a maze of size 17*17, and then randomly generate the initial positions of the hero (letter "H"), five monsters (letter "M"), and five goblins (letter "G") in the maze. The maze is composed of wall bricks (symbol "#") and routes (symbol "-" , and see the provided maze generator program for more details), and the maze is a closed one with no entrance and exit. A possible initial game configuration is as shown in Figure 1 (it is obvious that none of the characters, i.e. the hero, monsters, and goblins should be deployed on the wall bricks):

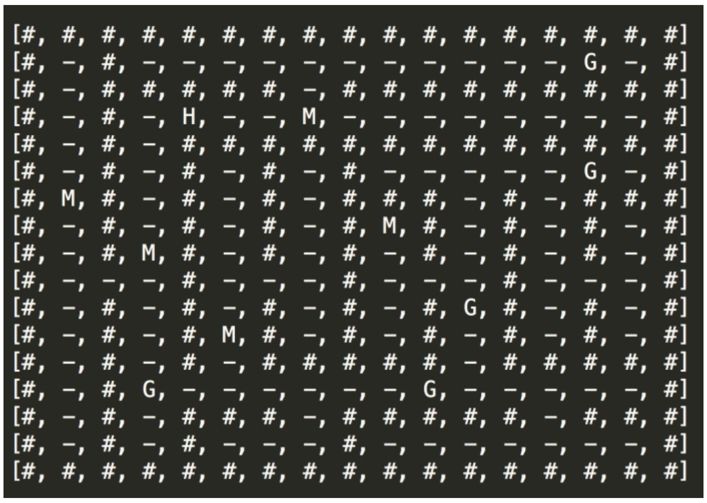


```
[#, #, #, #, #, #, #, #, #, #, #, #, #, #, #, #, #]
[#, -, #, -, -, -, -, -, -, -, -, -, -, -, G, -, #]
[#, -, #, #, #, #, #, -, #, #, #, #, #, #, #, #, #]
[#, -, #, H, -, -, -, M, -, -, -, -, -, -, -, -, #]
[#, -, #, -, #, #, #, #, #, #, #, #, #, #, #, #, #]
[#, -, #, -, #, -, #, -, #, -, -, -, -, -, G, -, #]
[#, M, #, -, #, -, #, -, #, #, #, -, #, -, #, #, #]
[#, -, #, -, #, -, #, -, #, M, #, -, #, -, #, -, #]
[#, -, #, M, #, -, #, -, #, -, #, -, #, -, #, -, #]
[#, -, -, -, #, -, -, -, #, -, -, -, #, -, -, -, #]
[#, -, #, -, #, -, #, -, #, -, #, G, #, -, #, -, #]
[#, -, #, -, #, M, #, -, #, -, #, -, #, -, #, -, #]
[#, -, #, -, #, -, #, #, #, #, #, -, #, #, #, #, #]
[#, -, #, G, -, -, -, -, -, -, G, -, -, -, -, -, #]
[#, -, #, -, #, #, #, -, #, #, #, #, #, -, #, #, #]
[#, -, #, -, #, -, -, -, #, -, -, -, -, -, -, -, #]
[#, #, #, #, #, #, #, #, #, #, #, #, #, #, #, #, #]
```

Figure 1: An Example Initial Game Configuration

The roles in the game are detailed below:

1. **<u>Hero</u>**: The hero is shown as the **letter "H"** in the maze, and there is only one hero in the game. The player controls the movement of the hero in four directions (*up, down, left, and right*), and each time the hero can only move one step along the maze route (symbol "-"). The hero cannot move across the wall ("#"), which means if they are asked to do so, they will stay where they are. Below is the result when the player presses the "right" arrow: the hero moves one step right from (Row 4, Column 4) as in Figure 1 to (Row 4, Column 5) in Figure 2:



```
[#, #, #, #, #, #, #, #, #, #, #, #, #, #, #, #, #]
[#, -, #, -, -, -, -, -, -, -, -, -, -, -, G, -, #]
[#, -, #, #, #, #, #, -, #, #, #, #, #, #, #, #, #]
[#, -, #, -, H, -, -, M, -, -, -, -, -, -, -, -, #]
[#, -, #, -, #, #, #, #, #, #, #, #, #, #, #, #, #]
[#, -, #, -, #, -, #, -, #, -, -, -, -, -, G, -, #]
[#, M, #, -, #, -, #, -, #, #, #, -, #, -, #, #, #]
[#, -, #, -, #, -, #, -, #, M, #, -, #, -, #, -, #]
[#, -, #, M, #, -, #, -, #, -, #, -, #, -, #, -, #]
[#, -, -, -, #, -, -, -, #, -, -, -, #, -, -, -, #]
[#, -, #, -, #, -, #, -, #, -, #, G, #, -, #, -, #]
[#, -, #, -, #, M, #, -, #, -, #, -, #, -, #, -, #]
[#, -, #, -, #, -, #, #, #, #, #, -, #, #, #, #, #]
[#, -, #, G, -, -, -, -, -, -, G, -, -, -, -, -, #]
[#, -, #, -, #, #, #, -, #, #, #, #, #, -, #, #, #]
[#, -, #, -, #, -, -, -, #, -, -, -, -, -, -, -, #]
[#, #, #, #, #, #, #, #, #, #, #, #, #, #, #, #, #]
```

Figure 2: The hero moves one step right.

   The hero has 100 health points and 1000 gold coins at the beginning of the game. Each movement of the hero consumes 1 health point. **The hero will die if their health points drop to zero, which also means the end of the game and no ranking will be recorded. The game is successful if the hero meets all the monsters at least once and still survives (health points>0).** The monsters will not disappear even when they lose a fight (see below), but goblins disappear once they interact with the hero. After a successful game, the coins remaining will be used to determine the rank of the player. So a player needs to strategically choose the route in order to get the best ranking.

2. **<u>Monsters</u>**: They are the hero's enemies, and they are randomly generated and deployed in the maze at the beginning of the game with varying abilities. They are shown as the **letter "M"** in the maze map. There are three types of monsters:

   a. *Thief Monsters:* they will steal gold coins from the hero, and depending on their ability, they steal a different number of gold coins with a certain success rate. For instance, one such monster may have the ability of (10, 90%), which means they can steal 10 coins from a hero with a success rate of 90%.
   b. *Fighter Monsters:* they like to fight with the hero. The health points of the hero may be reduced after fighting. For instance, one such monster may have the ability of (30, 40%), which means they will reduce the hero's health score by 30 with a probability of 40%.
   c. *Gamer Monsters*: they will play the rock-paper-scissors game with the hero. If the hero wins the game, they will get through the monster without any loss; otherwise, the hero will lose some coins and health points. One such monster may have the ability of (100, 50), which means if they win the game, they will get 100 coins from the hero and reduce the hero's health score by 50.

3. **<u>Goblins</u>**: They are the hero's friends, and they will either give coins to the hero or increase their health points. Goblins will be randomly generated and deployed at the beginning of the game with

varying abilities. They are shown as the **letter "G"** in the maze map. There are three types of goblins:

a. A *Wealth Goblin* will give a random amount of gold coins to the hero when they meet. One such goblin may have the ability of (100, 50%), which means they will give the hero 100 coins at a probability of 50%.
b. A *Health Goblin* will help recover the health of the hero by increasing the hero's health points. One such goblin may have the ability of (50, 70%), which means they may give 50 health to the hero at a probability of 70%.
c. A *Gamer Goblin* will play the same rock-paper-scissors game with the hero, and if the hero wins the game, they will be given a specified number of coins and health points. One such goblin may have the ability of (100, 50), which means if the hero wins the game, they will get 100 coins and 50 health points from the goblin.

Your Assignment:
You are required to implement the above game in Python 3 using appropriate OOP concepts and techniques (e.g. class, object, method, inheritance, encapsulation …). Some partially completed code has been provided for you to start with, and you are free to extend and modify the code.

You should print necessary well formatted text messages on the screen to inform the player about the game status at each step and provide an appropriate input interface for the player.

Finally, you need to write a short report (max 4 sides of A4) to describe your program and presenting the results of your game, including appropriate screenshots. See report details under 'Submission Requirements' below.

Evaluation:
Your submission will be marked according to the following criteria:

- CGS D:
  - The program is executable.
  - Game Initialisation: At the beginning of the game, the program will **randomly** generate a maze (17*17) with the hero, **five monsters**, and **five goblins** with **varying types and abilities**. It is noted that the initial positions of them should be random rather than fixed.
  - Report on Initialisation: The program will then report on screen the initial positions of the monsters and goblins and their individual types and abilities. **All three types of monsters and all three types of goblins should be generated**.
  - Report on available operations: If a user presses the "H" key, the program should output all the available operations (keys) a user can enter and their functions. In this way, a user does not have to refer to the game manual.
  - Hero Moving: When the player presses one of the four arrow keys (up, down, left, and right), the screen should display which key has been pressed. A new game map should be printed out on the screen to reflect the change (e.g., the hero being in the new position as in Figures 1 and 2).
  - Map Display: When the player presses the "M" key, the current map should be printed on the screen, including everyone's position in the maze.
  - Fight/interaction pre-information: When the hero meets the monsters or the goblins, necessary messages need to be displayed on the screen, including the type of the monsters/goblins encountered and their abilities. Print out plain English to explain what

will happen after the fight/interaction. One example is as follows: *"The hero meets a gamer goblin with the ability of (130, 40) at coordinate (4, 5). A game will be played and if the hero wins the game, they will get 130 coins and 40 health from the goblin."*

- CGS C:
  - The program fulfils the requirements for CGS D.
  - *Fight/Interaction Calculation:* When a hero meets a monster or a goblin, the program should calculate the outcome of the fight/interaction and update and report the hero's status. You should do this for all types of monsters and goblins.

    One example output could be: *"The hero fought with the fighter monster at coordinate (4, 5) and lost the fight, XX health points were lost, the current hero's status is (health: 920, coins: 350)".*

    Another example could be *"The hero fought with the fighter monster at coordinate (4, 5) and won the fight, the hero's status is: (health: 950, coins: 400)"*

  - Game Termination: You should determine when the game is terminated: either the hero is killed or the hero meets all monsters at least once. You should also output messages informing the player if a game is successful or not.
  - Playable Game: You should set up the abilities of the goblins and monsters in a way that the hero cannot survive for ever (say, more than 100 steps) or be killed in very few steps (say, in 2 to 3 steps).
  - Automatic rock-paper-scissors game: you can automatically play the game by randomly generating decisions for both the hero and the gamer goblin/monster and decide the winner. You should consider the draw situation (in which case another round needs to be run). You should report the game process to the user, including the decisions made by both parties and the outcome of the game.

- CGS B:
  - The program fulfils all the requirements for CGS C.
  - Output Format: The program should produce well formatted and concise text messages on the screen, informing the player about the game status and available actions at each step.
  - League Table: You should maintain a league table recording the ranking of all players and displaying the names of the top 10 players ranked by coins left (regardless of the difficulty levels).
  - Display Ranks: If a game is successful, you should ask the player to input their name and show their ranking based on the league table. You should also update the league table. You will then output the top 10 players' names and their performance at the end of the game.
  - Interactive rock-paper-scissors game: you should engage the player in an interactive session by asking the player to make a decision by pressing a single key (R for Rock, P for Paper, and I for Scissor). You should then inform them of the outcome of the game.

- CGS A:
  - The program fulfils all the requirements for CGS B.
  - OOP elements: The program should demonstrate excellent use of various OOP concepts as necessary.
  - Report and Program Quality: The report should be well written and comprehensive, and the program should be well written and documented.
  - Save the game: A player can save the game to a file by pressing the key "S". The player can then quit the program, restart the program, and reload the game by pressing the key "L". Note the file must be saved in the same folder as your source code file. The league table should also be saved and it can be reloaded after restarting the program.

- Difficulty Levels: The player can choose four difficulty levels to play the game: easy, medium, hard, and very hard. For different levels you should appropriately adjust the abilities of the monsters and goblins. This also means the league table should rank the players individually based on the difficulty levels.
- Better League Table: The league table should include the ranking for each difficulty level. At the end of the game, only the top 10 players at each difficulty level should be displayed.

Submission Requirements:

By the deadline of **Monday March 9th 2020 at 12:00 noon**, you are required to submit a single compressed archive (ZIP) file containing the following:

- **Mini-project report**
  A short report (Word document, max 4 sides of A4) containing the following:
  - Brief descriptions of the game you have implemented, including how to play the game and what classes you have used.
  - Examples of screenshots of the game to help users understand the game.
  - UML class diagrams for any classes you implemented in your solution.
- **Source code**
  Your complete Python code – suitably documented and organised into modules/packages.

You should name your ZIP file as follows:

```
<surname>_<studentID>.zip
```

If your surname was Einstein and your student ID was 51761234, then the filename would be:

```
einstein_51761234.zip
```

Upload your submission using the CS1527 Assessment/Mini-Project link in MyAberdeen.