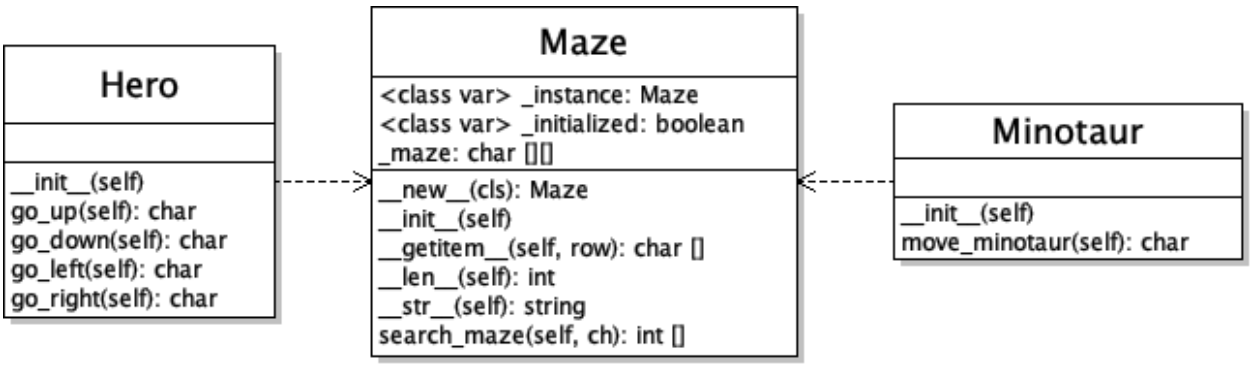
CECS 277 – Lab 9 – Singleton

Minotaur's Maze

Create a program that allows the user to wander through a maze that is guarded by a minotaur. The user wins if they reach the maze's exit without being captured by the minotaur. Use the following UML diagram and the class descriptions below to create your program:



Classes:

1. Maze – singleton – the minotaur's maze.
   a. __new__(cls) – if the maze hasn't been constructed, then construct it and store it in the instance class variable and return it. If it has, then just return the instance.
   b. __init__(self) – create and fill the 2D list from the file contents.
   c. __getitem__(self, row) – overloaded [] operator – returns the specified row from the maze. (Note: this operator can be used to access a row (ex. m[row]) or can be used to access a character at a row and column (ex. m[row][col])).
   d. __len__(self) – returns the number of rows in the maze. (ex. to get the number of rows, use: num_rows = len(m), for columns, use: num_cols = len(m[row])).
   e. __str__(self) – returns the maze as a string in a grid format.
   f. search_maze(self, ch) – returns the location of the character (ch) in the maze as a two-item 1D list (ex. [row, col]).
2. Hero – the player's character
   a. __init__(self) – sets the hero's starting location by finding the start position in the maze ('s') and then placing an 'H' in the maze at that location.
   b. go_up/go_down/go_left/go_right(self) – update the hero's location by finding the 'H' in the maze, overwriting it with a ' '. Then add or subtract 1 to either the row or col for the appropriate direction, check to make sure the new location is not a wall ('*'), if it isn't, overwrite the new location with an 'H'. Return the character that was at that location so that main knows if the user ran into the minotaur, the finish, or a wall.
3. Minotaur – the monster that guards the maze.
   a. __init__(self) – randomizes the starting location of the minotaur to any blank space in the maze. Place an 'M' at that spot.
   b. move_minotaur(self) – every turn the minotaur will detect the hero's location in the maze and then move one space toward that direction. Check that the space is not a wall ('*') or the finish ('f'), if it is, then choose a different direction. Overwrite the old 'M' with a

space and place a new ‘M’ at the updated position. Return the character that was at that location so that main can know whether the minotaur captured the hero.

Main – construct the Maze, Hero, and Minotaur objects. Display the maze and prompt the user to enter a direction to move the Hero in. Move the Hero, if the Hero reaches the finish, then the user wins, if the hero runs into the Minotaur, then they lose, if the Hero hits a wall or moves to an empty space, then the Minotaur moves. If the Minotaur captures the Hero, then the user loses the game.

Example Output:

```
* * * * * * * * * * * * * * * * * * * *
* H                       *             *
*   * * * *   *   * * *   *   *   * * * *
*   *       *       *       *       *   *
*   *   * * * *   *   * * *   *   *   *   *
*       *       *           *   *   M   *
* * *   *       *               *       *
*           * * *   *       *       *   *
*                   *       *   *   *   *
*   * * * * *       * * *   *   *   *   *
*               *               *   *   *
*   * * * * *   *   * * * * *   *       *
*               *                   f   *
* * * * * * * * * * * * * * * * * * * * *
Choose a Direction (WASD):x
Choose a Direction (WASD):d
```

```
* * * * * * * * * * * * * * * * * * * *
*     H                   *             *
*   * * * *   *   * * *   *   *   * * *   *
*   *       *       *       *       M   *
*   *   * * * *   *   * * *   *   *   *   *
*       *       *           *   *       *
* * *   *       *               *       *
*           * * *   *       *       *   *
*                   *       *   *   *   *
*   * * * * *       * * *   *   *   *   *
*               *               *   *   *
*   * * * * *   *   * * * * *   *       *
*               *                   f   *
* * * * * * * * * * * * * * * * * * * * *
Choose a Direction (WASD):d
...
```

```
* * * * * * * * * * * * * * * * * * * *
*   H                   *             *
*   * * * *   *   * * *   *   *   * * *   *
*   *       *       *       *       *   *
*   *   * * * *   *   * * *   *   *   *   *
*       *       *           *   *       *
* * *   *       *               *       *
*           * * *   *       *       *   *
*                   *       *   *   *   *
*   * * * * *       * * *   *   *   *   *
*               *               *   *   *
*   * * * * *   *   * * * * *   *       *
*               *                   f   *
* * * * * * * * * * * * * * * * * * * * *
Choose a Direction (WASD):w
You ran into a wall!
```

```
* * * * * * * * * * * * * * * * * * * *
*   H                   *             *
*   * * * *   *   * * *   *   *   * * *   *
*   *       *       *       *       *   *
*   *   * * * *   *   * * *   *   M *   *
*       *       *           *   *       *
* * *   *       *               *       *
*           * * *   *       *       *   *
*                   *       *   *   *   *
*   * * * * *       * * *   *   *   *   *
*               *               *   *   *
*   * * * * *   *   * * * * *   *       *
*               *                   f   *
* * * * * * * * * * * * * * * * * * * * *
Choose a Direction (WASD):d
```

```
* * * * * * * * * * * * * * * * * * * *
*                       *             *
*   * * * *   *   * * *   *   *   * * *   *
*   *       *       *       *       *   *
*   *   * * * *   *   * * *   *   *   *   *
*       *       *           *   *       *
* * *   *       *               *       *
*           * * *   *       *       *   *
*                   *       *   *   *   *
*   * * * * *       * * *   *   *   *   *
*               *               *   *   *
*   * * * * *   *   * * * * *   *       *
*               *                   M H f *
* * * * * * * * * * * * * * * * * * * * *
Choose a Direction (WASD):s
```

```
* * * * * * * * * * * * * * * * * * * *
*                       *             *
*   * * * *   *   * * *   *   *   * * *   *
*   *       *       *       *           *
*   *   * * * *   *   * * *   *   *       *
*       *       *           *   *   *   *
* * *   *   * * * *   *       *   *   *   *
*           *       *   *   *       *   *
*   *               *       *   *   *   *
*   * * * * * *   *   * * *   *   *   *   *
*       *       *               *   *   *
*   * * * * * *   *       *       * * *   *
*               *               M     H *
* * * * * * * * * * * * * * * * * * * * *
You found the exit!
```

Notes:

1. You should have 4 different files: main.py, minotaur.py, hero.py, maze.py.
2. Use docstrings to document each of the classes, their attributes, and their methods.
3. Place your names, date, and a brief description of the program in a comment block at the top of your main file. Place brief comments throughout your code.
4. Do not create any extra methods, parameters, attributes, or modify the class structure.
5. Please do not create any global variables (besides the singleton maze) or use attributes globally (ie. do not access any of the attributes using the underscores).
6. The Maze is a singleton, that means that whenever and wherever you need to use one of its methods, just construct an instance (which will always be the same instance).
7. The Minotaur's move_minotaur method can determine the movement in any way you'd like. You might want to start off by choosing a random direction (up, down, left, or right, as long as it isn't a wall), and then later, refining your method to track the user by searching for the 'H' in the maze and then heading in that location by using a few if statements. Hint: try not to let it get stuck on a wall, if its direction is into a wall, choose a random direction.
8. Thoroughly test your program before submitting:
    a. Make sure that the Maze class is a singleton so that only the one instance is ever used throughout the program.
    b. Make sure that the maze is read in correctly from the file and stored in the 2D list.
    c. Make sure that when you're reading in the maze it can work for any size maze (not just the one provided).
    d. Make sure that the Hero begins the game at the starting location ('s').
    e. Make sure that the Minotaur starts at a random space in the maze (' ').
    f. Make sure that the Minotaur and the Hero cannot move out of bounds of the maze or walk through walls. The Minotaur also cannot move onto the finish ('f').
    g. Make sure that the Hero can move in any direction within the maze.
    h. Make sure that when the Hero or Minotaur move, their old mark is removed and replaced with a space and a new mark is placed at their updated location.
    i. Make sure that the game ends when the finish is reached or when the Hero is captured (either by walking into the Minotaur, or the Minotaur walking into the hero).
    j. Make sure that the Minotaur moves every valid turn (ie. if the user walks into a wall, then the Minotaur moves, but if the user enters an invalid input, then it does not move).

Minotaur Maze Rubric – Time estimate: 4 hours

Minotaur Maze **10 points**	Correct. 2 points	A minor mistake. 1.5 points	A few mistakes. 1 point	Several mistakes. 0.5 points	No attempt. 0 points
Maze class (in a separate file)**:** 1. Singleton: has 2 class variables. 2. new checks if instantiated and always returns the same instance. 2. init method checks if initialized and reads in maze from file if not. 3. Overrides getitem, len, str. 4. Has search_maze method.					
Minotaur class (separate file)**:** 1. init places 'M' in maze at random blank spot in the maze. 2. move_minotaur method moves the minotaur once space toward the hero and returns the character that was at the location. Removes 'M' from old position and re-adds it at new position. 3. Does not move minotaur out of bounds, into wall, or the finish.					
Hero class (separate file)**:** 1. init places 'H' at start of maze ('s'). 2. goUp/Down/Left/Right methods moves the hero one space in the specified direction and returns the character that was at the location. Removes 'H' from old position and re-adds it at the new position. 3. Does not move hero into a wall.					
Main file (in separate file)**:** 1. Creates Maze, Hero, and Minotaur. 2. Repeatedly prompts user to move the Hero, then moves the minotaur and displays the updated maze. 3. Error checks user input. 4. Ends game when user finds the finish or is captured by the minotaur.					
Code Formatting: 1. Correct spacing. 2. Meaningful variable names. 3. No exceptions thrown. 4. No global variables (other than singleton) or uses attributes directly. 5. Correct documentation.					