Technical Description

The maze is a rectangle containing at most 1000 rows and 1000 columns. Each row is a String and all rows have the same length. Each cell (i.e., the intersection between a row and a column) of the maze can be either

* An empty space (represented as the space ' ' character)
* A wall (represented as the dot '.' character)
* The Exit gate (represented as the 'X' character)

There is exactly one Exit gate in the whole maze. You can be sure that it's always possible for the robot to reach the Exit gate eventually. You can also be sure that the maze will have walls around the four edges such that it's impossible for the robot to fall out of the maze.

In the picture below, there is a maze of 30 rows and 100 columns. If your robot starts at the top-left, it can follow the red path to reach the Exit gate. If your robot starts at the bottom-left, it can follow the blue path to reach the Exit gate. Note that those paths are not unique. There are much more paths to reach the Exit gate.

Take a look at this sample maze

.....  
.   X  
. .  
.....

The above maze has four rows and five columns. Let's assume that the robot is positioned initially at row = 2 and column = 1 (zero-based index). From the initial location, the robot can reach the Exit gate by following this: UP, RIGHT, RIGHT, RIGHT. Another possible path is RIGHT, RIGHT, UP, RIGHT. There are other non-optimal paths as well. For example, this path is not optimal: DOWN, DOWN, DOWN, DOWN, DOWN, UP, RIGHT, RIGHT, RIGHT (in this example, the robot is quite stubborn - it should stop going "DOWN" after the first unsuccessful try).

Note that your robot has NO information about the maze. As such, it must try different directions and mark walls accordingly.

You need to create a class Robot. The class contains one required public method:

void navigate()

In this navigate() method, your code must:

First, create a new Maze instance (I will provide a sample Maze implementation - you don't need to do anything)

[Sample Maze and Robot implementationLinks to an external site.](https://github.com/TriDang/cosc2658-2022-s3/tree/main/project_real)

Then, repeatedly call go() method of the Maze instance. For each call, you must provide one of the four valid values: "UP", "DOWN", "LEFT", "RIGHT" as the parameter. The go() method returns either "false", "true", or "win". Your code must process the returned values to know how to proceed with the subsequent calls to go().

Your code must stop whenever it receives "win" from a call to the go() method.

You must create your own data structure to keep track of the visited locations.

**More information about testing your program**

The following steps will be used to test your program

* A maze is generated (your robot will NOT know anything about this)
* The location of your robot is assigned (your robot will NOT know anything about this)
* An instance of your Robot class is created
* The navigate() method of the Robot instance will be called
* Anytime your robot calls to the go() method of Maze, a counter is increased
* The robot must try to make this counter as small as possible when it reaches the Exit gate

Deliverables

You need to provide me with the following outputs regarding the system you develop

* A README.txt file that describes the contribution score of all members (explained in the Contribution Score section below) and contains the link to the project video (explained in the Video Demonstration section below)
* Source code: Java source code (no library included) - Or Python source code (for master's students only) (similar to the requirements of Java program, Python program can only use built-in types and functions)
* Technical report: a Word/PDF document that contains the following sections
  + An overview and high-level design of your system (Java/Python classes, methods, their relationships, software design patterns, etc.)
  + The data structures and algorithms you apply or develop (note: you are not restricted to the data structures and algorithms covered in this course; you can create your own data structures and algorithms). You need to describe in detail the working of your data structures and algorithms.
  + Complexity analysis: you need to provide the complexity analysis of the algorithms you proposed.
  + Evaluation: you need to describe how you evaluate the correctness and efficiency of your system
* Video demonstration: create a short video (less than 15 minutes) to show a demo of your system. You have to upload your video to Youtube/OneDrive and present its URL in the REAME.txt file (your video must be accessible from any RMIT account).

Contribution Score