MI.103 Programming Techniques
Project – *Part 1*

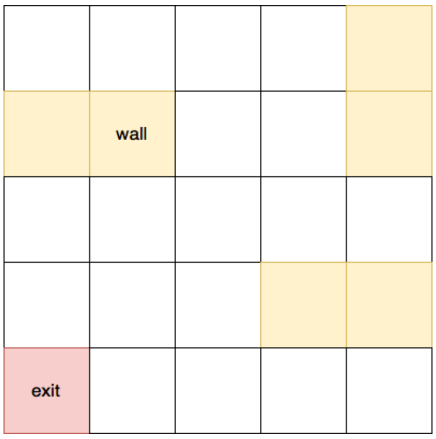
The goal of this project is to set up a labyrinth game with stupid robots (or crazy) and smarter robots. For example, a robot can move in any direction and can also go around in circles. It is also possible to imagine smarter robots with elaborate strategies to find it faster. You use exception to catch unexpected behaviour for each robot.

Exercise 1: The abstract class Maze

A maze is represented by a 2D grid of cells. Each cell has a position and a boolean indicating if this cell is a wall. A maze also contains an output and a list of robots that move in the 2D grid. A maze contains a pure virtual method *generateWalls ()* to generate the walls. Implement the abstract class *Maze*.

Exercise 2: The concrete class RandomMaze

A random maze is a maze with random walls. This class inherite of the abstract class Maze and contain two attributes :
— the number of the walls ;
— their maximal length.
For example, with three walls and a maximal length of two, you must obtain the follow result :



Implement the concrete class RandomMaze. You must override the pure virtual function *generateWalls()* and test it into a main program.

Exercise 3: The abstract class Robot

A robot is defined by a start position and a list of covered position. A robot has a method *go()*. This method depends on the robot's strategy to move.

Exercise 4: The crazy robot

This class models a crazy robot and inherits the Robot class. You must replace the pure virtual function *go()* to simulate the movement of the robot. This robot can go in all directions but the displacement is of a cell.

Exercise 5: The left robot

This class model a robot that always goes on the left. Implement the *LeftRobot* class.

Exercise 6: The right, up or down robot

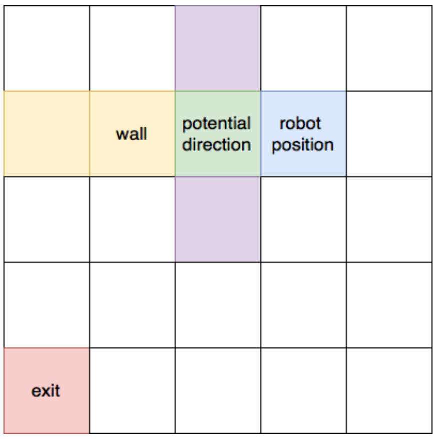
In the same way as the previous exercise, implement classes *RightRobot*, *UpRobot*, *DownRobot*.

Exercise 7: The jumping robot

This class models a jumper robot. A jumping robot chooses a random direction and can skip a wall or multiple cells (the number of skipped cells is an attribute of the robot). Implements the *JumpingRobot* class.

Exercise 8: The smart robot

This class models a smart robot. A smart robot evaluates the solution of each potential direction and chooses the best direction. It has a score function for all potential solutions. For example, consider the following situation :



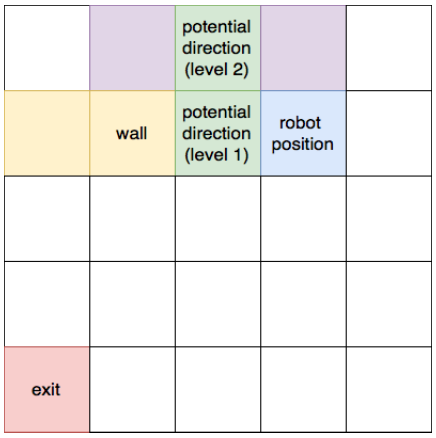
The robot evaluates the f^s score function for the left direction. This function can depend on the number of free cells C in the neighborhood of the cell :

$$f^s = \sum_j^4 C_j$$

We obtain $f^s = 2$ for the left direction.
In the same way, the robot evaluates the score function in all directions (right, up and down) and chooses the direction that has the maximum score function. If several directions have a maximum score function (same value), the robot chooses a direction in these directions. Implement the SmartRobot class.

Exercise 9: The very smart robot

The depth of this search can also be an attribute of the smart robot. For example, if multiple directions have a maximum score function, it is possible to evaluate the score function of all neighboring positions for each potential direction, as in the following figure :



It is necessary to use a tree to store each score function in each direction, each node can represent a parent direction with a score function and each leaf can represent neighboring cells with their score. This tree can be model into a class.

Implement the classes *ScoreTree* and VerySmartRobot.

Exercise 10: Maze test

Implement a main program to test all the robots and their movement. You can compare the mumber of covered cells for each of them.

Exercise 11: Complexe maze

Imagine and implement a maze using another wall generation strategy, more complexe - for example with walls in a circular way). Try to compare the behavior of the smart robot with a complexe maze and a random maze.