**Artificial intelligence :**

**Robot in a maze**

As an assignment, we had to create à robot which can find its path in a maze. The robot is able to rotate and go in every directions (up, down, left right). The text file given represents à maze written with “X” for walls and spaces for path.

We chose C++ as programing language.

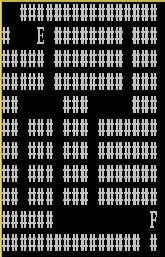
List of classes we used (see the annex for the code):

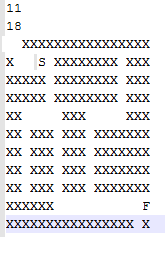
**Maze**: The maze class we implemented represents the maze given in the file. To keep it simple we implemented our maze with a matrix of boxes (the Box class is described below) represented by a vector of vector of boxes (std::vector<std::vector<Box\*> >).We used pointer of boxes, otherwise the program doesn’t work.

Moreover the maze has as attribute a robot and a Pledge robot (we will describe it later).

To create an instance of the maze, we just have to read the text file, and associate each character read with the correct box/robot, ‘X’ for a wall (in the console, walls are represented by #), ‘ ‘ for a passage, ‘S’ for a robot and ‘F’ for an exit. This part (read and create the maze) wasn’t complicated at all and to our mind it wasn’t the main purpose of the assignment, so we decided to omit it in this rapport.

The only important thing to mention is that we had to modify the text file, adding the number of line and column at the top.

For instance:

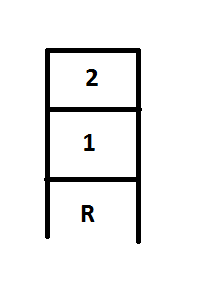


The maze class has a few methods like displaying the maze in the console, adding a box to the maze (used at the creation of the instance of maze), displaying the moves of the robot and the total cost.

**Robot:** This is the most important class of the program because it contains the algorithm of the movement.

First we decided to make a simple algorithm: the robot won’t rotate (because it can moves in every direction and the rotation is more expensive) and it will try to go right first, if it can’t it will try down, and then left and finally up. To know if the robot can move, it simply look at the box next him (for example if the robot has (x,y) as coordinates, the box below it in the matrix will have (x, y+1)as coordinates. So the robot just has to check if the box is a passage and not a wall to move into it). When the robot moves, we add in a list of char the letter of the movement (‘U’ for up etc)

To be sure that the robot visits the entire maze, we added to the robot a list of box (std::vector< \* Box>) in which every boxes the robot has visited are added into this list. So the robot will tend to go to unvisited boxes first.

But there is a problem with this list, if the robot enters in a dead end, it won’t be able to go back because it is not allowed to go to visited boxes. So we implemented a Return method by adding to the robot a stack of Boxes (std::stack<\* Box>). Like the previous list, every visited box is added to the stack. Let see with a simple image:

When the robot goes into the box number 1, the coordinates are added to both the list and the stack. Then when it goes to the box number 2 the coordinates are added too, but the robot won’t be able to go back to the box number 1 because the list already contains the coordinates of it.

So, to move back, we can use the stack getting the coordinates of the box number 1 (doing stack.pop and returning stack.top). Like this the robot will move to the box number 1 and at the next step it won’t be able to go to the box number 2 because it’s a visited box (the coordinates are still in the list).

The last task is to add the letter of the movement in the list, to do that we compare the current box and the box in which the robot return, if it’s above, that’s means the robot is going Up etc.

This algorithm is quite simple but the robot is able to find the exit, but after some researches we find an algorithm more complicated but more efficient to solve the path finding problem, it’s called the Pledge algorithm.

So we tried to implement it but it was more complicated than the first algorithm.

**Pledge Robot:** This class inherits of the Robot class and it’s called RobotP. This robot is able to rotate and the orientation is taken into account, moreover the robot will only go forward (implicitly the robot can move in every direction, it just has to rotate before going forward). So the robot can be oriented north, south east or west. The orientation is implemented by a simple int, when it’s equal to 0 the robot faces north, when it’s 1 the robot faces east etc.

To sum up, if the robot(x,y) is facing north and goes forward, it will got to the Box (x,y-1), and if it’s facing east it will go to the box (x+1,y).

In this algorithm we use a counter in order to know which direction the robot should choose. If the counter is equal to 0 the robot must go forward until it encounters a wall.

Then if the counter is lower or equal to 0 and if there is a wall in front of the robot, the robot will try to turn right first.

And if the counter is superior to 0 the robot will try to turn left.

When the robot turns right we decrease the counter, and when it turns left we increase it.

With this algorithm the robot is able to find a path to the exit.

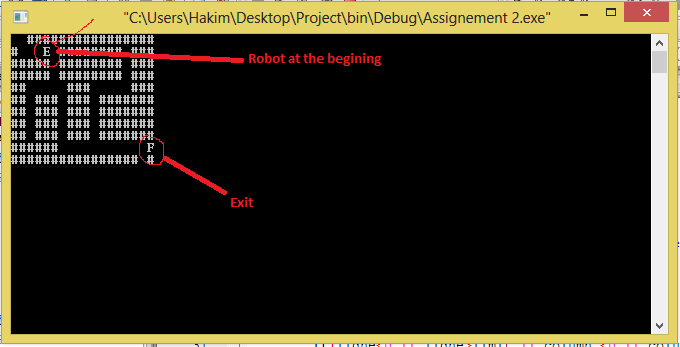
**Box:** The box class is a very simple class, a box is represented by a position as (x,y) coordinates. The box class is a virtual class.

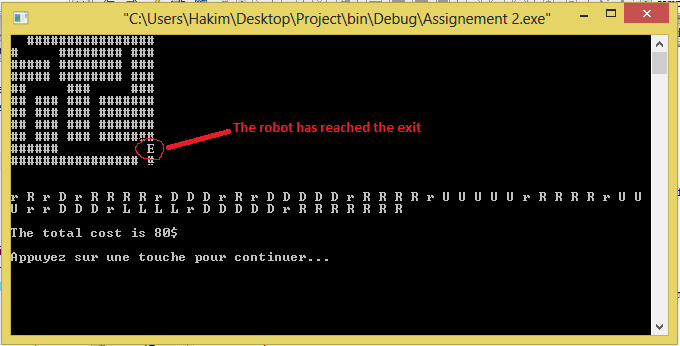
So we created two derived classes, Wall and Passage. The main differences are, first that a wall will be represented by ‘#’ in the console, and a passage either by ‘ ‘, or ‘F’ for an exit, or ‘r’ for a robot (there is a specific case for the Pledge robot). Moreover, a wall is never available for a robot, and a passage can be an exit or not.

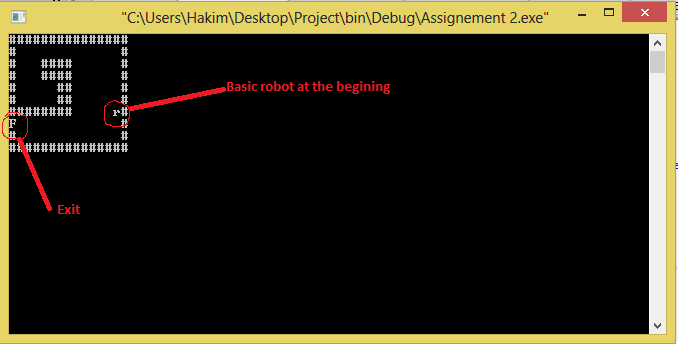
In this assignment we tried two algorithms. The first one is a very simple algorithm and the other one is more complicated. The difference between the two can be seen by trying more than one maze.

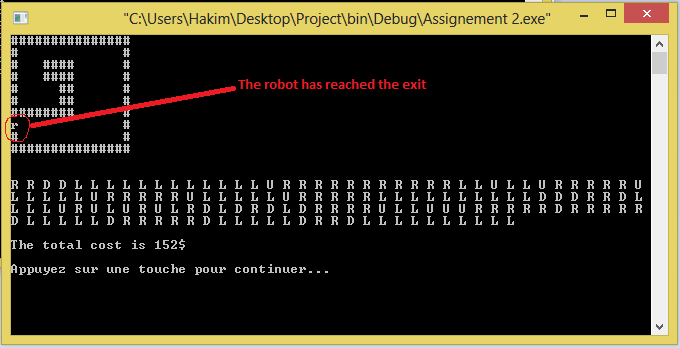
In fact, for smaller mazes, the simple algorithm is more efficient because the robot won’t rotate (a rotation cost 2$). But if we create a bigger maze (with larger passage) the first algorithm becomes less efficient because the robot will tend to visit every boxes of the maze.

Here are some examples of the execution of the program:



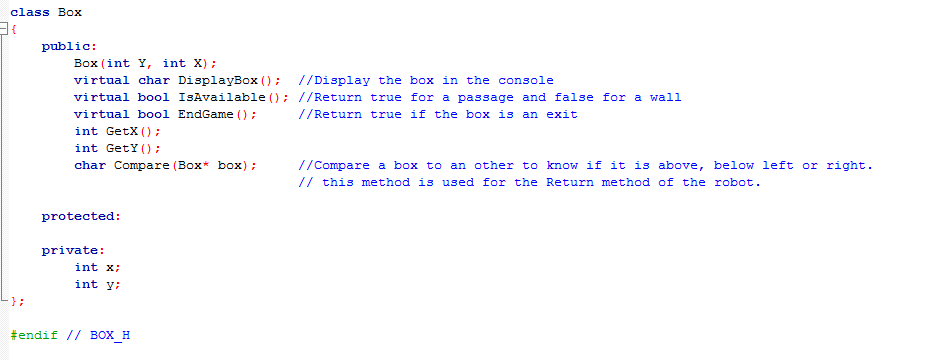




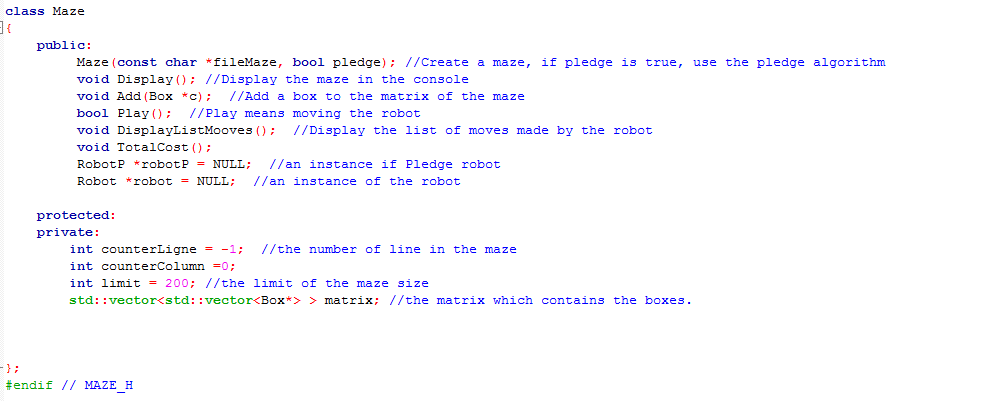


And here are the main parts of the code:

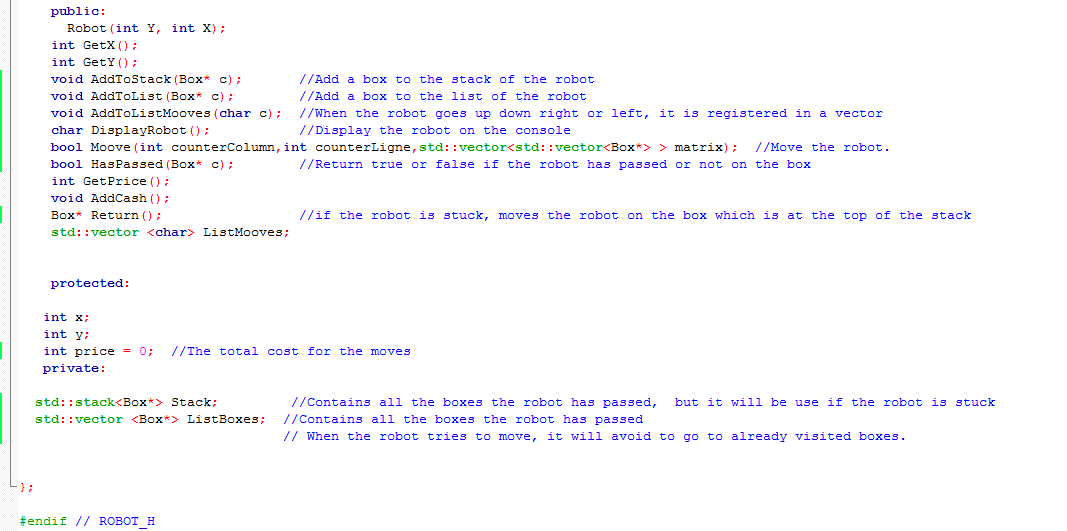
**Box:**

****

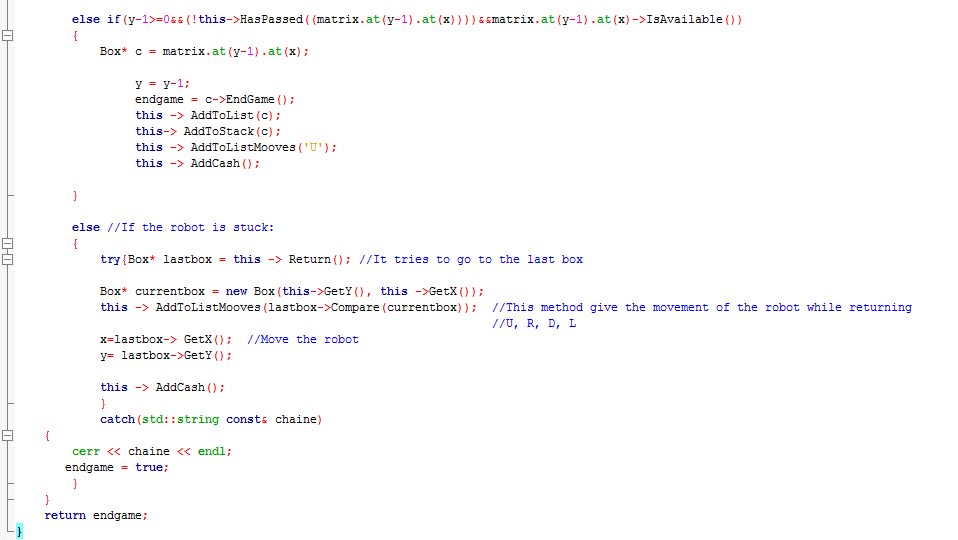
**Maze:**

****

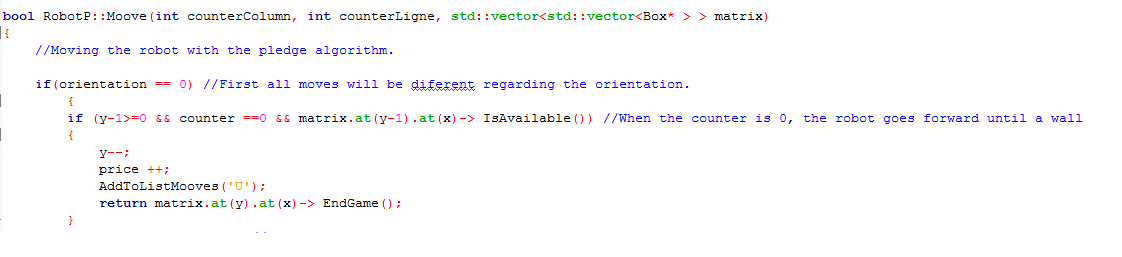
**Robot.h:**

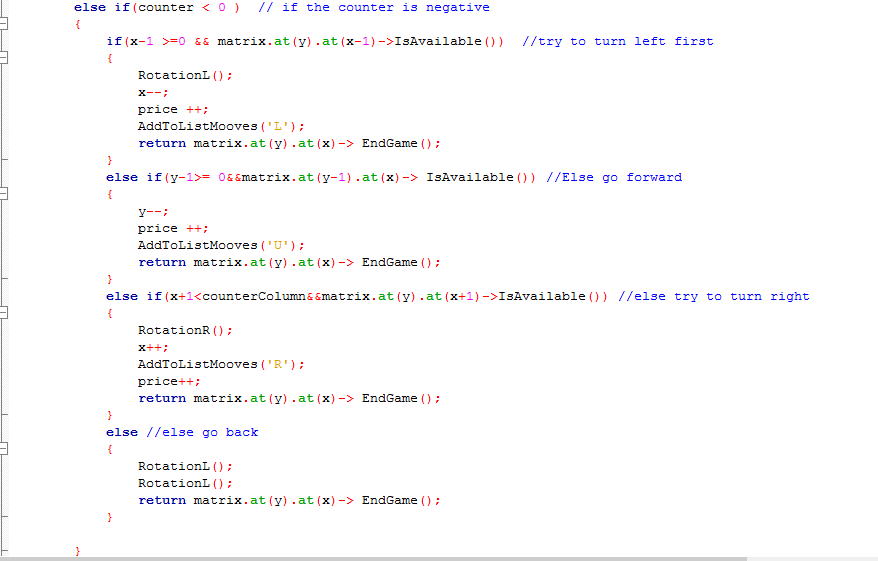
****

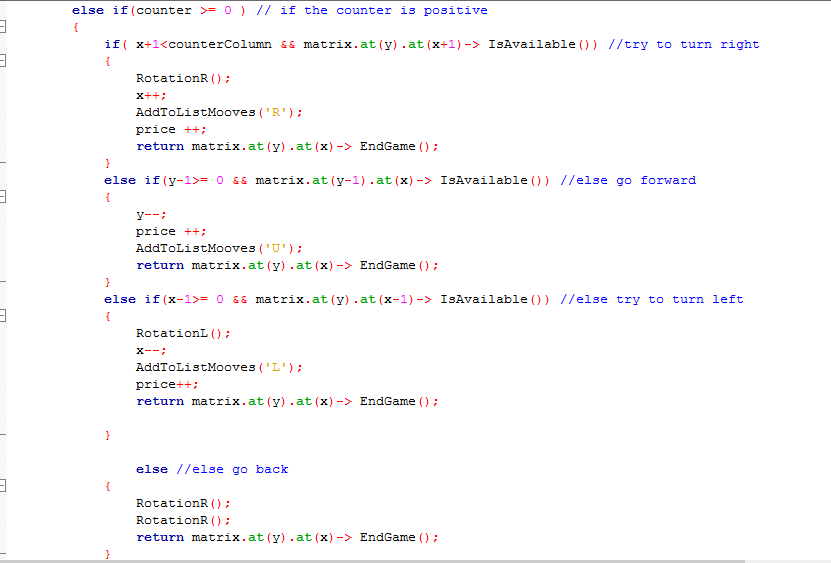
**First algorithm (only one direction):**

****

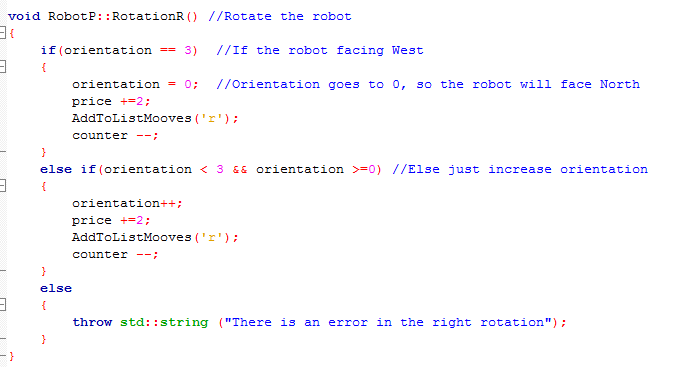
**Second algorithm (only one orientation):**

****

****

****

**Rotation:**

****