Team: Lorenza, Diego Calatayud, Cecilia Garduño

Teacher: Eduardo Serna Duarte

Algorithm: Find a solution to the maze.

```
Input:        0   1  2   3  4   5
      0      [['x', 'x', 'x', 'x', 'x', 'x'],
      1      ['x', ' ', ' ', ' ', 'S', 'x'],
      2      ['x', ' ', 'x', 'x', ' ', 'x'],
      3      ['x', ' ', 'x', 'x', ' ', 'x'],
      4      ['x', ' ', 'x', 'x', ' ', 'x'],
      5      ['x', ' ', 'x', ' ', ' ', 'x'],
      6      ['x', ' ', 'x', ' ', 'x', 'x'],
      7      ['x', ' ', ' ', 'E', ' ', 'x'],
      8      ['x', 'x', 'x', 'x', 'x', 'x']]
```

Output: Best solution to maze so that variable “S” = [currentLocationX][currentLocationY]

1. First, we have to build the function to verify if the position we are in is next to the exit (“S”).
2. Define checkIfNextToSolution (matrix, ycoordinate, xcoordinate)
3. If (ycoordinate, xcoordinate) is next to S then return True
4. Else return false
5. Define the matrix of Labyrinth through the variable named “maze”.
    a. Maze’s matrix= [rows 0-8][columns 0-5]
6. Assign your current location the values of (0,0) and create a cycle to move until the coordinates of currentLocation= “E”
7. Define “E” coordinates as E=[currentlocationY][currentLocationX]
8. Now we can start solving the maze by checking all possibilities.
9. Check possibilities of movement through commands that verify if movements up, down and side to side are possible.
10. While
    a. If maze[currentLocationy][currentLocationx-1] == “ “:
        i. PathLeft= false.
    b. If maze[currentLocationy+1][currentLocationx] == “ “:
        i. PathUp= true.
    c. And so on, and so forth.
    d. Print a dot each time the path is true a
    e. Add 1 to the number of paths/steps it took to get out of the maze

11. Now the function should return True and the maze is solved
12. Print a phrase to indicate you solved the maze. Ex: “you got out of the maze”
13. Print the number of paths or steps it took to solve it to make sure you found the optimal solution.
14. End of the Algorithm