Computer Programming 2018-2019
Unofficial Draft
Assignment: Maze

Floor van Helsdingen & Guus Vlaskamp

November 15, 2018

1 Introduction

The aim of this project is to code a robot which will search for the exit of a given maze. You will be given a maze with a number of properties. An exit and a starting point, and on which tiles the robot can walk and on which it cannot. Your robot will start at the starting position and autonomously 'walk' until it exits the maze. To this end you will write the code for this robot, which applies an algorithm to find the exit. This algorithm is not allowed to use the entire path visited by the robot, it only has limited memory so the robot can only remember the amount of times it went a certain direction, not the entirety of its path.

2 Assignment

We start with creating an object which creates the environment the robot can walk through; we will call this object the `Maze` class. Most importantly, this is the object that we will be constructing from a .txt file, to test your work. You can find an example of such a file in the elaboration; note that correct use of the formatting of such an input file is paramount here.

(a) **Create the `Maze` class, which represents the Maze the robot can walk through. It should at least contain the following:**

  i. **An `__init__()` that reads the input file and sets the following concepts:**

    I. **Create a two-dimensional array according to the given characters, where the elements of the array exist of one of the following integers:**
      - **0, which represent a tile that the robot can walk**
      - **1, which represent an obstacle in the labyrinth**
      - **2, which represent the starting location of the robot**

    II. **Create an attribute that contains a list of 2 integers, which represents the finish of the labyrinth. The finish can be recognized by the 0 at the edge of the maze, being the only tile on the edge of the labyrinth that the robot can walk on.**

    III. **Create an object of the class `Robot` which takes as input the starting position. The starting position is an array of two numbers. (You will not need these for a while. (for now, it suffices to simply store the object.)**

  ii. **A method of the form `get_finish(self)` which returns the location of the finish of the maze.**

Now that we have the `Maze` class, we will create a `Robot` class.

(b) **Create the `Robot` class, which represents the robot which can walk through the maze. It should at least contain the following:**

  i. **An `__init__()` method which should at least contain the following properties:**

    I. **An attribute that contains a list of 2 integers which represents the current position of the Robot.**

    II. **An attribute that contains a list of 2 integers which represents the last position of the robot. When creating a robot, set the last position equal to the current position of the robot. This attribute does not have to be given in the input. Later it can be changed by a method.**

    III. **An attribute that contains an object of the class `Maze`.**

    IV. **An attribute which consist the last direction of the Robot.**

  ii. **A method `change_pos(self, pos)` which will change the last position of the robot into the current position, and change the current position into the position indicated by the input of the method.**

  iii. **A method `walk_options(self)` which returns a dictionary of the options where the robot can walk to from its current position. An example of the return is:** *'left': [1, 1], 'right': [1, 3], 'down': [2, 2]*

  iv. **A method which will change the last direction of the Robot.**

  v. **A method `is_finished` which returns True if the Robot is at the finish and False otherwise.**

Next, we would like to work towards some methods that will choose the steps of the robot. Naturally, the real challenge is to search for solutions how the robot can walk in the labyrinth intelligently. This is where you will have full creative freedom. However, there exist several automated algorithms for solving mazes. You can also follow one of those. A few examples of these algorithms are:

- Wall follower (also known as the left-hand rule of the right-hand rule)
- Pledge algorithm
- Trémaux's algorighm
- Dead-end filling
- Recursive algorithm
- Maze-routing algorithm
- Shortest path algorithm

Create in the `Robot` class a method `take_one_step(robot)`, with which the robot chooses and takes a step, and returns the coordinates of the tile to which it moves. You can invent a method yourself or use an existing algorithm.

Finally, we are ready to put the final gear into place, and to fire up our program.

(e) **Create a `main()` function which creates a labyrinth and robot. Next, let the robot walk through the labyrinth until it is at the finish. The function needs to return the path of the robot.**

3 Elaboration

Testcase 1

```
11111111111
10000001001
11011101101
12010100101
10110110001
10000001101
10110111100
10010000011
11111111111
```

Draft 1: Format of the testcase files

Testcase 2

```
111111
100000
101011
102011
101011
100011
111111
```

Draft 2: Format of the testcase files

Main code

```
def main():
    maze1 = Maze('maze1.txt')
    maze2 = Maze('maze2.txt')
    robot1 = maze1.robot
    robot2 = maze2.robot
    path1=[]
    path2=[]

    while i<100 and (robot1.is_finished() == False or robot2.is_finished() == False):
        path1.append(robot1.take_one_step())
        path2.append(robot2.take_one_step())
        i+=1
```

Format of the main() in which your code will be ran to check.