OOP

Robot in a Maze

1 Introduction

Providing a robot with navigation ability through a territory is an interesting challenge in the field of Artificial Intelligence. In reality, the robot would be equipped with some sensors and tracking components, yet the underlying navigation algorithm is the most important ingredient for a computer scientist.

In this coursework we consider the simple example of a robot which tries to find its way out of a maze. A maze consists of pathways and walls and the challenge is to find one's way out of such a structure.

Figure 1 Longleat maze in England

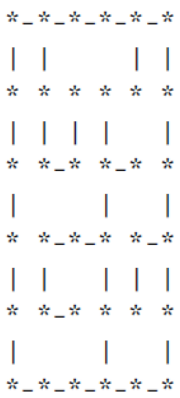
You are required to write a program that can load the information about a maze from a file, and then simulate a few types of robots navigating their way out of the given maze.

Note that the focus is on your Object Oriented Design rather than any decorations of the program, and that is how the coursework will be judged.

2 Maze files

You are provided with a few text files each representing a maze. An example of a 5-by-5 maze represented in text format is given in Fig 2 below.

Figure 2 A (5 × 5) maze represented in text format



To create a maze object, you need to read the representation in these files. In doing so, you need to observe the following:

1. Each cell in a pathway is represented as a space character, i. e. ' '. Note that the space character is di_erent from any other whitespace character (such as a tab, new line, etc).
We make the following convention: a space represents a pathway cell. Any other character should be regarded as a wall cell. For simplicity, in our files we only use '*', '-' and '|' for walls.
2. As you see from Fig. 2, a 5-by-5 maze is represented by an 11-by-11 matrix of characters. In fact, counting the new line character at the end, on a Linux system it would be an 11-by-12 matrix, and on some other systems that represent the new line by two characters, an 11-by-13 matrix. So, be careful with the new line at the end of each row when you read from these files.
It is not di_cult to see that, ignoring the new line characters, an m-by-n maze requires a (2m + 1)-by-(2n + 1) matrix of characters.
3. A maze can have many entrances and exits. Again for simplicity we only consider those mazes that have their one entrance at the top left and their only exit at the bottom right. This means that under two dimensional array indexing, the entrance to the maze in Fig. 2 is at index (0,0) and the exit is at index (4,4).
For a general m-by-n maze, the only entrance is at (0,0) and the exit is at (m-1,n-1).

3 Object Oriented Design

A simple search on the Internet shows that there are many programs available online which implement the main task of the current coursework. But our focus is mainly on the design of the program. You are required to apply the di_erent concepts from Object Oriented Programming that you have learnt in this module to accomplish the task.

3.1 Classes

Try to think of all the di_erent elements of the program in terms of objects of di_erent class types. Some of these elements are listed below. Note that these are just suggestions:

1. Maze.
2. Robot.
3. Position: This could be:

--- the position of a robot in a maze
---the location of a junction in a maze
---the location of the entrance
---. . .

4. Game manager: One piece of the program should handle each 'game'. The game manager takes a maze and a robot, and registers the maze with the robot, and then asks the robot to make successive moves, and after each move checks to see whether the move has been a legal move or the robot is just cheating (e. g. moving through a wall).

In reality, such a robot would be monitored by the game manager using tracking devices. But here we keep it simple and just obtain the position of the robot through its methods.

5. Other entities: You need to come up with the rest yourself.

You need to work out what class type matches each of these concepts best. For instance, would it be better to have a Robot interface which is implemented by each specific robot, or would an abstract class provide a better solution?

3.2 Exceptions

You should modularise your program in such a way that other people can use its components independently. This means that when designing (say) a class, you should not assume that all the constructors and methods would be called carefully by other people as you will not have control over how others use the class.

Thus, you need to handle careless usage gracefully through the proper use of Exceptions. You need to come up with a few exception classes custom designed for the current task. Whenever a third party uses any of your methods carelessly, they should be provided by informative and appropriate exceptions.

4 Animation

Ideally one would show the movements of a robot in a maze through a well designed graphical interface. As we have not covered graphics in our module you are only expected to provide a text based output.

For this, you have at least two options:
1. A more pleasant approach is to print each frame of the animation on the console, then pause for a short time (say, 200 milliseconds), then clear the screen and move on to the next frame. The problem with this approach is that there is not a uniform way of clearing the console in Java.
For instance, the following statement:

```
System.out.println("\u001b[2J");
System.out.flush();
```

clears the screen if the console supports ANSI codes. This includes the following:
---The usual terminal on a Linux system.
--- The usual cygwin and MinGW consoles on MS Windows.

but does not include:
_ the MS Windows command prompt.
_ The terminal in an IDE such as Netbeans or Eclipse.
2. Another less elegant approach is to just print subsequent frames one after another, separated by some line. For instance, the following shows the first move of a robot from cell (0,0) to (0,1):

```
Robot type -> RandomMemoryRobot
move number -> 0

*-*-*-*-*
|R    | |
*-*  * * *
|   |   |
*  *-*-*  *
|      | |
*-*-*  * *
|      | |
*-*-*-*-*
----------------------------
Robot type -> RandomMemoryRobot
move number -> 1

*-*-*-*-*
|  R  | |
*-*  * * *
|   |   |
*  *-*-*  *
|      | |
*-*-*  * *
|      | |
*-*-*-*-*
----------------------------
...
```

5 Robots

You must implement four types of robots:
Right hand robot: This robot always turns right at any junction.
Left hand robot: This robot always turns left at any junction.
Random robot: At any junction, this robot chooses its direction randomly.
Intelligent robot: You should design the strategy for a robot which behaves a bit more intelligently than the previous ones. For instance, one suggestion is to to have a combination of randomness and 'memory', so that whenever the robot reaches a dead end, it remembers not to go down that route anymore.
Of course this is just a suggestion.
The exact algorithm employed is up to you. This is where you should use your own creativity.

6 Reports

Once finished with the implementation, in addition to your source code, you are required to submit two reports in .pdf format:

Class design and the usage instructions: (at most 2 pages) You should explain your package and class design in about one page. It is preferred if you include a diagram of how your classes are related to each other. Note that this does not mean that you can compromise the clarity of your code. You should try to write your code as clear as possible, so that ideally I do not feel the need to read this report.

Draw a UML diagram if you know what they are. Otherwise, a simple graphical representation would su_ce and you would not lose any marks if you do not use UML diagrams.

You should explain your design choices. For instance, you should explain why you choose (say) an interface rather than an abstract class or a normal class (or vice versa) for any of the particular entities in the program. You should say what extra functionality this choice provides and/or explain convincingly why conceptually your choice makes sense.

Finally, you must mention:

6

_ which class is the entry point to your program, i. e. which class includes the main method.

_ how and from which directory I should issue the running command. You should write the full command, for example:

```
directory: build/classes
command: java testMaze.TestMaze
```

In case your program accepts any command line arguments, you should provide a list of those options. For example, your program may accept a command line argument specifying which type of robot the user would want to try, as in:

```
directory: build/classes
command: java testMaze.TestMaze RightHand
```

Finally, in case your program does not provide choices through an interface, you should explain how the user can try your program with di_erent types of robots and mazes. You may even specify which lines in your main class need to be changed (say, by commenting out and uncommenting) for this to happen.

Note that this report altogether should not exceed 2 pages.

Intelligent robot: (at most 1 page) You should write a brief account of how your intelligent robot behaves and why you think your implementation is a good one. You may provide an argument based on how the robot performs on

average compared with the other types of robots in this coursework. Ideally one would provide a mathematical proof, but here the results of some experiments would be enough.
This part can be a mere paragraph, but in any case it must not exceed one page.

7 How to submit

Compress the source of your program in zip, tar or jar format (with the source files included inside the jar file) and submit together with your two reports via Moodle. You must observe the following:
7
1. You must keep the package (folder) structure in the compressed file, otherwise your program will not compile.
2. Make sure you send the source files (i. e. the .java files), and do not just send the class files by mistake. This is especially important if you compress your files in jar format. If you forget to send the source files, you will lose all the marks.
3. Your reports should be written clearly, and must not exceed the page limits given. They are meant to be used as references for me.

8 The marking scheme

The marking scheme is as follows:
Correctness: (20%) Your program should work correctly. In particular, if your program cannot be compiled and run with a standard JDK, you will lose all the marks for correctness.
Design: (30%) You should design your program properly with regards to Object Oriented principles and concepts. You should explain your choices in your report as well.
Style: (20%) You should observe all the tips that have been given to you in the book, lectures, labs and tutorials regarding your style of programming, e. g.:
_ Informative identifier names;
_ Proper use of UPPER CASE, lower case, CamelCase and camelCase styles for the identifiers in your code;
_ Succinct but adequate comments;
_ Proper use of blank lines to separate various sections of the code;
_ Correct indentation;
_ etc . . .
The rule of thumb is, I should be able to read your program easily. If your code is badly written, depending on how bad it is, you may lose all the marks for style.
Intelligent robot: (20%) For this you need to come up with your own idea. Your implementation must be correct, i. e. your intelligent robot must always find its way out of the maze, otherwise it would not be that intelligent after all.
8

Moreover, you should explain your idea clearly in your report about the intelligent robot.
Reports: (10%) Your reports should complement your implementation. The user should be able to understand the main design you have employed and how to use your program with the help of these reports.

9 Important points to consider

You should consider the following points very carefully.

9.1 Maze file format and the Operating System

The maze files have been generated on Linux. Hence, there is a possibility that the new line character in the file may be di_erent on your operating system if it is not Linux.
Nonetheless, no matter which operating system you use, your program must be able to reconstruct a maze from any of the given maze files.

9.2 Losing marks

Failing to observe any of the following would result in losing marks:
Compilation and execution: I will run your programs with the standard JDK 8. This means that you can use the latest features introduced in Java.
However, if you use any non-standard features that prevent your program from compiling or running smoothly on a standard Java compiler, you may lose all the marks.
Information about the maze: You may not assume that you have access to the information about the whole maze. In other words, at any moment, the robot only has access to the following information:
1. its own position (i. e. where it is)
2. in which direction it can take its immediate next move without hitting a wall.

Note that if you get information about the whole maze, then there are very fast algorithms that can take the robot to the exit. If you design your intelligent robot based on this assumption, then you will lose the mark for that.