Semester 2 2018
COMP3702/7702 ARTIFICIAL INTELLIGENCE
ASSIGNMENT 1: Search for Movers

Note:

- This assignment consists of two parts: Programming and report.
- You can do this assignment in a group of **at most** 3 students. This means you can also do the assignment individually.
- For those who choose to work in a group:
    - All students in the group must be enrolled in the same course code, i.e., all COMP3702 students or all COMP7702 students.
    - Please register your group name in http://goo.gl/bgAaCZ before **5pm on Friday, 24 August 2018**. If you have not registered your group by the said time, you will need to work on the assignment individually.
    - All group members are expected to work in both programming and report. The demo will involve Q&A to each group member, individually.
- Submission Instruction:
    - Your program should compile from command prompt and generate an executable that can be run from command prompt as:

      ```
      > java ProgramName inputFileName outputFileName
      ```

    - You should submit **only the source code** required to compile the program, i.e., remove all object files and executable before submission.
    - The report should be in .pdf format and named a1-[courseCode]-[ID].pdf.
      If you work individually, ID is your student number.
      If you work in a group, ID is the student number of all group members separated by a dash. For instance, if you work in a group of two, and the student number is 12345 and 45678, then
      [ID] should be replaced with 12345-45678
    - The report and all the source codes necessary to compile your program should be placed inside a folder named a1-[courseCode]-[ID].
    - The folder should be zipped under the same name, i.e., a1-[courseCode]-[ID].zip, and the zip file should be submitted via turnitin before **5pm on Friday, 14 September 2018 with weekend grace period**.
    - Weekend grace period means there will be no late penalty for submissions before 8am on Monday, 17 September 2018.
- Demo Instruction:
    - Please register for a demo slot in http://goo.gl/WrgDP4 before **5pm on Friday, 31 August 2018**.
    - If you work in a group, all group members must be present for the demo.
    - You must use the lab PC for the demo, and therefore you must make sure that your code does compile and run well in the lab PC.
- **Amendment:**
    - 18 Aug 2018: Please amend the width of movable obstacles from in the range $[0.5w, 1.5w]$ to in the range $[w, 1.5w]$.

The many building refurbishment at UQ has caused the emergence of a new business opportunity: Movers!!! Of course UQPF would like to get a piece of this business. For this purpose, they hire you to develop a robot that can move boxes around the building. This project is your attempt to create a simplified prototype for such a robot. For this prototype, you will only need to develop the software that will allow a robot to push 2D moving boxes from one location to another. Subsequent information provides details of the problems and assumptions for building this prototype.

The robot is represented as a line segment of length w unit, where $w \in [0.05, 0.15]$. The robot operates in a 2D environment of size [0, 1] X [0, 1] populated by moving boxes, movable obstacles, and static obstacles that cannot be moved. All moving boxes are axis-aligned squares with size wXw unit^2. All movable obstacles are axis-aligned squares whose width is in the range [~~0.5~~w, 1.5w]. All static obstacles are axis-aligned rectangles. Throughout this assignment, we *define the (x, y) position* of the robot, a moving box or a movable obstacle to refer to *the position of the center point of the respective line segment and squares*, respectively.

The robot's motion has the following constraints:

1. The robot has 3 DOFs: Its configuration is determined by position (x, y) of the center point of the line segment and orientation α. Figure 1 illustrates the robot in its operating environment.
2. The robot can push only one moving box or one movable obstacle at a time.
3. To push a moving box or a movable obstacle, the robot must first place itself, such that at least three-quarter of its length coincides with a side of the box/obstacle. Once in this position, the box/obstacle will follow the motion of the robot as long as at least three-quarter of the robot's length coincides with the side of the box/obstacle. If the robot does not satisfy this requirement, the robot and the box/obstacle will remain where it is.
   Note: There is no separate action to trigger pushing of the box/obstacle.
4. Each primitive action moves the robot by a distance of at most 0.001 unit.

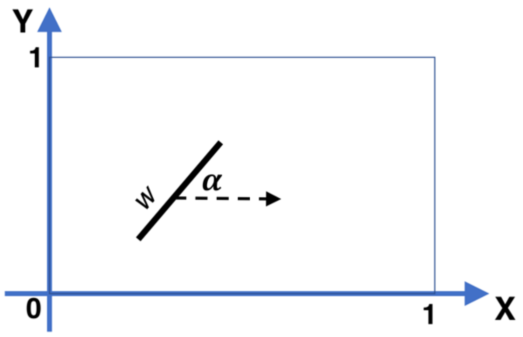

Figure 1. Illustration of the robot in an empty environment.

Your task is to develop a program that outputs a collision-free path for the robot to push the moving boxes to their respective goal locations. Collision-free path means, the path that the robot, each moving box, and each movable obstacle follow does not collide with any static obstacles in the environment, nor with each other.

We will provide a supporting code in Java that will help:

1. Parse the input file.
2. Visualize the scenario and path.
3. Check the validity of a solution.

These codes will be released on 5pm, Friday 24 August 2018.

Hint: Defining the problem is half the solution. Please use the one week (17 August to 24 August 2018) to work on the agent design problem and design your search solution. Please use the four questions in the report to guide your design.

Input format

The input of the program is a text file containing information on the robot and problem scenario. In particular, the file follows the following format:

- The first line consists of 4 numbers, where the first number is the value of w, while the subsequent two numbers represent the initial (x, y) position and the last number represents the orientation $\alpha \in [0, 2\pi]$rad.
- The second line consists of 3 numbers separated by a single white space. The numbers represent the number of moving boxes (m), the number of movable obstacles (n), and the number of static obstacles (o), respectively.
- Each line-*i*, where $i \in [3, m+2]$ consists of 4 numbers separated by a single white space, representing the initial and goal positions of the (i-2)th moving box. The first two number in line-*i* is the initial (x, y) position of the (i-2)th moving box, while the last two numbers is the goal (x, y) position.
- Each line-*i*, where $i \in [m+3, m+n+2]$ consists of 3 numbers separated by a single white space, representing the position and size of movable obstacles. In particular, the first two numbers represent the initial (x, y) position of the (i-m-2)th movable obstacle, while the last number represent the width of the obstacle.
- Each line-*i*, where $i \in [m+n+3, m+n+o+2]$ consists of 4 numbers separated by a single white space, representing the position and geometry of the static obstacles. The first two numbers represent the (x, y) position of the lower left vertex of the obstacle, while the last two represents the (x, y) position of the upper right vertex of the obstacle.

Example of an input file:

```
0.1 0.1 0.5 0.0
2 1 1
0.15 0.15 0.8 0.8
0.25 0.25 0.8 0.9
0.4 0.4 0.15
0.0 0.8 0.3 1.0
```

The input file means:

- The robot length is 0.1 unit, its initial (x, y) position is at (0.1, 0.5), while its initial orientation is α=0rad.
- The environment is populated by two moving boxes (each with width 0.1 unit), a movable obstacle, and a static obstacle.
- The initial position of the first moving box is (0.15, 0.15) and the goal is (0.8, 0.8).
- The initial position of the first moving box is (0.25, 0.25) and the goal is (0.8, 0.9).
- The initial position of the only movable obstacle is (0.4, 0.4), while its width is 0.15 unit.
- The only static obstacle is a rectangle, whose lower left vertex is at (0, 0.8) and upper right vertex is at (0.3, 1.0)

Output format

The output of the program is a text file containing the path. In particular, the file consists of p+2 lines, where p is the number of primitive steps in the path. The first line is the value p. Each subsequent line consists of 3+2(m+n) numbers separated by a single white space. The first three numbers are the configuration (x, y, α) of the robot, while the subsequent m pairs of numbers represent the (x, y) position of the moving boxes, and the last n pairs of number represent the (x, y) position of the movable obstacles. The order of the moving boxes and movable obstacles must be the same as the order in the input file.

Example of an output file for the above example input file:

```
300
0.1 0.5 0.0 0.15 0.15 0.25 0.25 0.4 0.4
0.101 0.5 0.0 0.15 0.15 0.25 0.25 0.4 0.4
:
0.78 0.825 0.0 0.8 0.8 0.8 0.9 0.4 0.4
```

The above output means the initial configuration of the robot is (0.1, 0.5, 0.0), while the initial positions of the moving boxes and movable obstacles are (0.15, 0.15), (0.25, 0.25), and (0.4, 0.4), respectively. In the first step, the robot moves from (0.1, 0.5, 0.0) to (0.101, 0.5, 0.0), while the boxes and movable obstacles are not moving yet. The final configuration of the robot and movable obstacles are (0.78, 0.825, 0.0) and (0.4, 0.4), while the goal position of the moving boxes are (0.8, 0.8) and (0.8, 0.9).

Grading for the Programming Part (total points: 60/100)

The details of the grading scheme is as follows. If your situation satisfies more than one marking band, we will use the higher band.

COMP3702:

- >= 1 & < 10: The program does not compile nor run (staff discretion).
- >= 10 & < 20: The program runs but fails to solve any query within the given time limit (staff discretion).
- 30: The program solves a query involving up to 2 moving boxes, no movable obstacles, and up to 2 static obstacles with planning time less than 1 minute.
- 40: The program solves a query involving 3-4 moving boxes, 2 movable obstacles, and up to 4 static obstacles (the obstacles are set relatively sparse) with planning time less than 1 minute.
- 50: The program solves a query involving 3-4 moving boxes, 2 movable obstacles, and up to 8 static obstacles with planning time less than 1.5 minute.
- 60: The program solves a query involving 5-7 moving boxes, 3-5 movable obstacles, and up to 10 static obstacles with planning time less than 2 minutes.

COMP7702:

- >= 1 & < 5: The program does not compile nor run (staff discretion).
- >= 5 & < 10: The program runs but fails to solve any query within the given time limit (staff discretion).
- 20: The program solves a query involving up to 2 moving boxes, no movable obstacles, and up to 2 static obstacles with planning time less than 1 minute.
- 30: The program solves a query involving 3-4 moving boxes, 2 movable obstacles, and up to 4 static obstacles (the obstacles are set relatively sparse) with planning time less than 1 minute.
- 40: The program solves a query involving 3-4 moving boxes, 2 movable obstacles, and up to 8 static obstacles with planning time less than 1.5 minute.
- 50: The program solves a query involving 5-7 moving boxes, 3-5 movable obstacles, and up to 10 static obstacles with planning time less than 2 minutes.
- 60: The program solves a query involving 8-10 moving boxes, 6-8 movable obstacles, and up to 12 static obstacles with planning time less than 2 minutes.

Note: The planning time is the total planning time. If you use on-line method, the total planning time means the total accummulated planning time over all steps.

Report (total points: 40/100)

Your report must contain answers to the following questions:

1. [5 pts] Please define the agent design problem for this movers prototype.
2. [10 pts] Please describe your search method at the conceptual level (i.e., pseudo code and what abstract data structure is used for the container). If you use sampling-based method, please describe the strategy you apply or develop for each of its four components. Otherwise, please describe the details of your discretization method.
3. [12.5 pts] Which class of scenarios do you think your program will be able to solve well? Please explain your answer.
4. [12.5 pts] Under what situation do you think your program will fail? Please explain your answer.

Please note that in each question, when explanation is requested, the explanation part is 80% of the total points. A good explanation should include a logical explanation of why and include experimental results to back your explanation. Also, please also note that good explanation is NOT equal to long explanation!!!

oOo That's All, Folks oOo