CSCI 561 Foundation of Artificial Intelligence Spring 2019
Homework 3

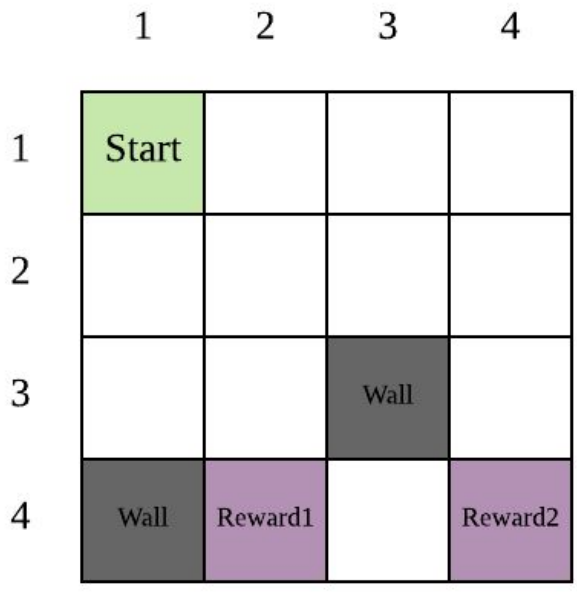
Due April 15, 2019 23:59:59

Introduction

"Cash Miner" is a popular flash game on the Internet, in which you control a virtual figure to move in a grid map and finally reach cells containing cash rewards. Recently the video game company "Universe of Soda Consumer" released a new version of the game "Cash Miner" called "Advanced Cash Miner", in which the virtual figure's are specified by the player before the game starts. The company also claimed that anyone who can crack the game and give the optimal control sequence to get the maximum expected rewards would receive a huge prize. As an outstanding student in the class of Foundations of Artificial Intelligence, you definitely would not want to miss such a good opportunity to show your ability. Please try to crack the game and win the prize.

Problem Description

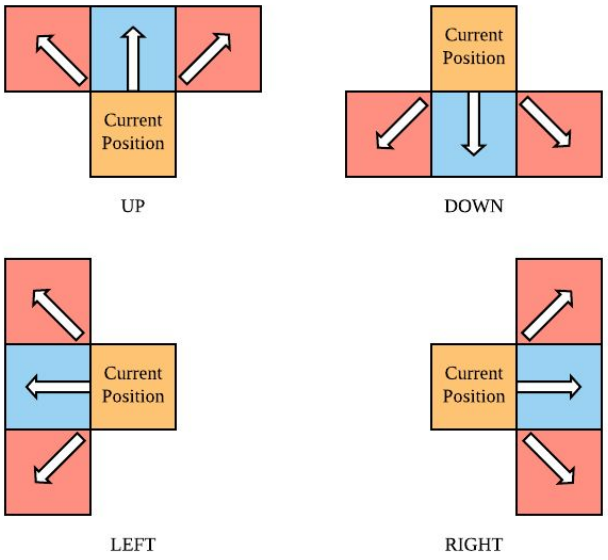
The virtual figure moves in a grid of size N * N, which means there are N * N square cells in the grid. There is a given cell or cells which contain(s) the cash rewards. In addition, there may be some cells containing walls. All other cells in the grid are empty. A typical grid is shown below:



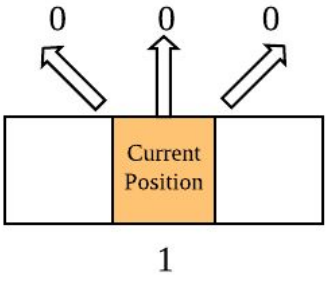
In the above example grid, the virtual figure's current position is at square (1,1). There are two walls in this grid (squares (3,3) and (4,1)). There are two reward squares in this grid: (4,2) and (4,4).

At each non-reward state, the virtual figure has four possible moves: Up, Down, Left, or Right. At the terminal state, there is only one possible action: Exit.

Unfortunately, the environment in this game is stochastic, because sometimes wind may blow the virtual figure away from its intended direction. Whenever the virtual figure moves in a direction to an empty square, it successfully reaches that square with probability *P*. However, there is also a probability of (1 - *P*)/2 that the figure will be blown off course either 45 degrees clockwise or 45 degrees counter-clockwise.

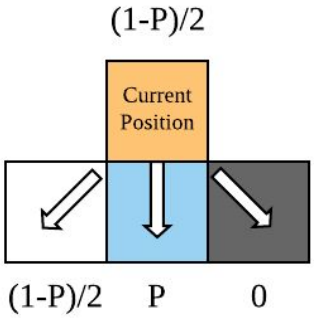


If the result of an action would move the virtual figure out of the grid, the virtual figure stays in its original cell. In the image below, the virtual figure attempts to move up from the top row in the grid, resulting in it remaining in its current position with probability 1:



Stay in the same postion!

If the result of an action would move the virtual figure into a wall cell, the virtual figure also stays in its original cell. In the image below, the virtual figure attempts to move down, but there is a wall in the bottom right square surrounding it. Because there is a probability (1-P)/2 that the wind blows the figure into that wall, there is a (1-P)/2 probability that the figure ends up back in the current position:



No way to bump into the wall

On every turn, the virtual figure receives a reward R(s, a). For all non-reward squares, the reward for any movement action is:

$$R(s) = Rp$$

When performing the Exit action in a state with reward, R_t, the corresponding reward is:

$$R(s) = Rt$$

You are willing to play this game as long as it takes to get your money. On the other hand, you want to receive your money as soon as possible, so money received on your next turn is not always worth as much as money received right now, by a discount factor of γ ($0 \leq \gamma \leq 1$).
Given a grid, please determine the optimal move sequence for the virtual figure to follow from any square on the grid to maximize your winnings.

Input Format
You should read into the input parameters from a text file called “input.txt”. The file would be formatted as below:
<GRID SIZE> The first line contains a single integer N, indicating the size of the grid.
<WALL STATES NUMBER> This line contains one integer M, indicating the number of wall cells in the grid.
<WALL STATES POSITION> The next M lines here are the coordinates of wall cells, each occupying one line. Each line has 2 integers separated by a comma “,” indicating the coordinates of the walls (row first, then column).
<TERMINAL STATES NUMBER> This line contains one integer T, indicating the number of terminal cells in the grid.
<TERMINAL STATES POSITION> The next T lines are the coordinates of terminal cells, each occupies one line. Each line has 3 integers separated by a comma “,”. The first two are the coordinates of the terminals (row first, then column), and the third value denotes the reward, Rt, received when exiting this square.
<TRANSITION MODEL> This line contains one floating-point number P, indicating the probability of moving as intended.
<REWARDS> This line contains one floating-point number indicating Rp.
<DISCOUNT FACTOR> This line contains a floating-point number γ in the interval [0,1] .

Output Format
<OPTIMAL ACTION GRID> The output file should contain N lines where each line includes N characters separated by commas (“,”). The j-th character at the i-th line should indicate the optimal action (one of “U”, “D”, “L” or “R” for non-reward states; “E” for reward states; “N” for wall states) at square (i, j).

- U: move up
- D: move down
- L: move left

- R: move right
- E: exit with the reward
- N: no actions in the wall state

Sample Input

```
5
2
4,2
1,4
2
5,3,10.0
3,5,5.0
0.8
-0.3
0.7
```

Sample Output

```
D,D,D,N,D
R,D,D,D,D
R,R,D,D,E
D,N,D,D,L
R,R,E,L,L
```

Grading

Your homework submission will be tested as follows: Your program should take no command-line arguments. It should read a text file called “input.txt” in the current directory that contains a problem definition. It should write a file “output.txt” with your solution. The formats for files “input.txt” and “output.txt” are specified above. End-of-line convention is Unix (because Vocareum is a Unix system).

During grading, 50 test cases will be used. If your homework submission passes all 50 test cases, you receive 100 points. For each test case that fails, you will lose 2 points.

Note that if your code does not compile, or somehow fails to load and parse input.txt, or writes an incorrectly formatted output.txt, or no output.txt at all, or OuTpUt.TxT, you will get zero points.

Homework rules

1. You must use Python 2.7 to implement your homework assignment. You are allowed to use standard libraries only. You have to implement any other functions or methods by yourself.
2. You need to create a file named “hw3cs561s2018.py”. When you submit the homework on vocareum, the following commands will be executed:
   Python hw3cs561s2019.py
3. Your code must create a file named “output.txt” and print its output there. For each test case, the grading script will put an “input.txt” file in your work folder, runs your program and check the “output.txt” file generated by your code. The grading script will replace the files automatically, so you do NOT need to do anything for that part.
4. Homework should be submitted through Vocareum. Homework submitted through email, in person during office hours, or through any channel other than Vocareum will not be accepted. Please only upload your code to the “/work” directory. Don’t create any subfolder or upload any other files. Please refer to http://help.vocareum.com/article/30-getting-started-students to get started with Vocareum.
5. Your program should handle all test cases within a reasonable time. A maximum runtime of **30 seconds** per test case will be set on Vocareum.
6. You are strongly recommended to submit at least two hours ahead of the deadline, as the submission system around the deadline might be slow and possibly cause late submission. Late submissions will not be graded.
7. **DO NOT PRINT ANYTHING TO THE CONSOLE**. This means no uses of the **print()** function should be in your submitted solution. If you do, it can cause slow down your program and cause Vocareum to time out during grading.