SI 335 Spring 2013: Project 3

Your name:

Due: Thursday, April 4

Instructions: Review the course honor policy: informal *discussions* are permitted, but no *written collaboration* of any kind..

This cover sheet must be the front page of what you hand in. Fill out the left column in the table to the right before you hand it in.

Use separate paper for your solutions and make sure they are submitted in order — please!.

Comments or suggestions about this problem set:

Comments or suggestions about the course so far:

I had discussions with:

Citations (be specific about websites):

Grading rubric:

- **5**: Solution is completely correct, concisely presented, and neatly written.
- **4**: The solution is mostly correct, but one or two minor details were missed, or the presentation could be more concise.
- **3**: The main idea is correct, but there are some significant mistakes. The presentation is somewhat sloppy or confused.
- **2**: A complete effort was made, but the result is mostly incorrect. There may be some basic misunderstandings of the topic or the problem.
- **1**: The beginning of an attempt was made, but the work is clearly incomplete.
- **0**: Not submitted.

Problem	Self-assessment	Final assessment
1		
2		
3		
4		
5		
6		

This project contains only electronic parts. Your code must be submitted according to the instructions on this page. No cover sheet is required, as there is no written component.

Coding Guidelines:

- It should be clear what your code is doing. The best way is to use sensible variable/function/class names, and to clearly state what every non-trivial chunk of your code (function, long loop, long block) does.
- Your program should read from standard in and write to standard out. Its I/O behavior should be exactly identical to the sample starter code. *You should write any error or debugging messages to stderr (cerr)*, as this stream will be ignored in testing.
- Follow the course honor policy: document any external sources, and don't produce or look at any code in collaboration with any classmate.

1 A Mazing Project

In this project, you are going to find optimal pathways through a maze. A maze will be represented by a text file of space characters, X's, and O's. The X's are walls, and the O's are "coins". For example, here's a 9-by-15 maze:

```
XXXXXXXXXXXXXXXXX
       X O      O X
XXX XXXXX X X X X
X O   X X   X XOX
XXXXX XOX XXX X X
X     X X X    OX
XXXXX XOX XXXXX X
X           X   X
XXXOXXXXXXXXX X X
X       OX O   X
XXXXXXXXXXXXXXXXX
```

The input to your program will be a maze like that one, piped into standard in. Your program will read in this maze and produce a sequence of "moves" starting at the top-left corner that will produce the maximum score, hopefully ending up at the bottom-right corner. Each move is either `U`, `D`, `R`, or `L`, corresponding to each of those four directions. And don't worry; I've written the I/O stuff for you, so all you need to write is a single function which takes in a 2-dimensional array that represents the maze, and returns a list of moves.

2 Awesome graphical interface

For your convenience, I have made a program to show this process a little more graphically for you. Here is how the maze above looks if you save it as `maze.txt` run

`python3 showmaze.py maze.txt`:

You can see that the blue diamond represents the current position. This maze can now be solved manually, with the points updated as you go. To analyze the performance of your program it will be more useful to run an existing file of moves through the given maze.

Here is a list of moves to solve that maze; probably not the optimal solution, but a fine one nonetheless:

```
R R R D D
R R D D D D
R R R R
U U U U U U
L R R R R R R R
DDDDDDDD R
```

(Note that whitespace in this output does not matter.) The moves above might be the output from your maze solving program. If the text above were saved as `moves.txt`, and you ran

`python3 showmaze.py moves.txt`

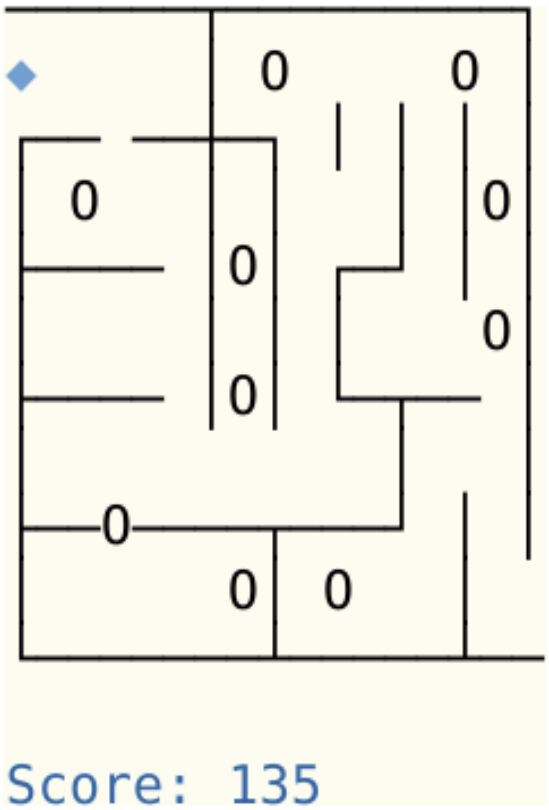

Figure 1: Sample maze

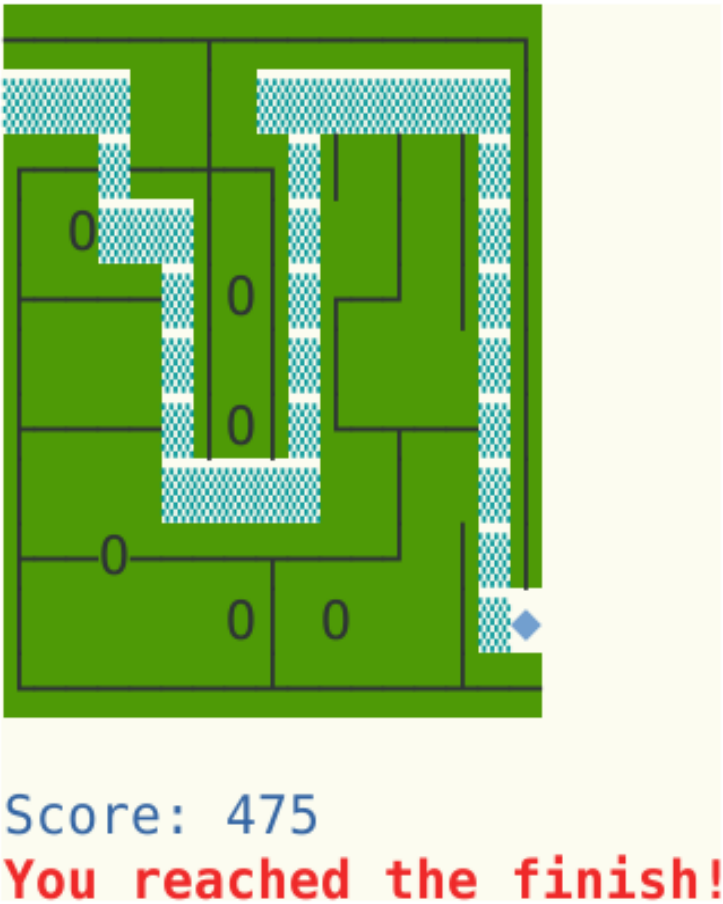

Figure 2: Sample maze with moves

then you would see an exciting animation, which concludes with

This shows the path of all the moves, with the blue diamond in the correct final position and having earned a hefty sum of points.

3 Points

The goal of your program is to solve the maze and earn the mazimal number of points in the process. Here is how points are awarded, for a maze with height h and width w:

- You start with wh points.
- Every move decreases your point total by 1.
- Picking up a `O` coin earns you $\lfloor wh/5 \rfloor$ points.
- Arriving at the bottom-right corner finishes the maze and earns $2wh$ points.
- If you run into a wall, go out of bounds, or run out of moves, the maze is over and your point total is whatever it is at that moment.

So for example, in the 9 by 15 maze above, the starting point total shown is $9 \cdot 15 = 135$ points. The series of moves contains a total of 38 movements, which finds 4 coins, and the finish line. So the final point total is

$$135 - 38 + 4 \cdot 27 + 270 = 475$$

4 Other helper programs

There are 2 more Python programs I've written to help you develop and debug your code:

- `genmaze.py`: This program generates random mazes for you. Run `python3 genmaze.py` without any arguments to get a run-down of how the program is used. For instance, the example above was generated by running
  `python3 genmaze.py 15 9 rPrims`
- `scoremaze.py`: This runs a set of moves through a given maze and produces the final score, without any of the graphical niceties. This is how your programs will be auto-tested. You just run
  `python3 scoremaze.py maze.txt moves.txt`
  (where `maze.txt` and `moves.txt` are text files that store the maze description and list of moves, respectively), and the output is some informational messages, followed by the final total score for that list of moves.

5 Starter Code

You can get all these files at once by downloading proj3.tar.gz and running `tar xzvf proj3.tar.gz`

- solvemaze.py
- solvemaze.cpp
- solvemaze.java
- genmaze.py
- showmaze.py
- scoremaze.py

The only thing you need to write is the `calcMoves()` method in solvemaze.XXX. You can use Python, C++, or Java, as you like. Please don't modify the helper scripts `genmaze.py`, `showmaze.py`, or `scoremaze.py`.

6 Grading

Your program will be tested on a variety of maze files ranging from the smallest possible mazes like 10x1 and 3x4 mazes, all the way up to a 1000x1000 maze. The test mazes will vary in size as well as their structure (how many walls they have). However, you are promised that:

- The start will always be in the second row, first column.

- The end will always be in the second-to-last row, last column.
- There will always be at least one path from the start to the end.
- There will always be exactly 10 coins.

This is how the `genmaze.py` generated mazes work, so those would be good examples to use for testing.

I will only run your solvemaze program, which should output a list of moves for the given input maze. You should only submit *one* `solvemaze.XXX` file, depending on the programming language you have chosen to implement your program.

Your program will be allowed a maximum of 3 seconds on each imput maze file. After this, your program will be terminated and any moves that have been output so far are the ones that will be tested.

After running your program on each maze, I will use the `scoremaze.py` program, along with that maze file, to find out how many points your "moves" score. Say your moves score x points. After looking at all of the sumbissions, as well as my sample solutions, I will also get the maximum number of points any of them score on that maze file, which I will call m. Your grade for that maze will be determined from the ratio $\frac{x}{m}$ of your points divided by the maximum number of points for that maze. There will be bonus points available *on every sample maze*, if you get the maximum number of points (or close to it).

As usual, besides these auto-tests, your program will also be graded on readability, documentation, and code organization. Follow the "coding guidelines" outlined at the beginning of this page.