CSCI 136 Assignment – Mazes

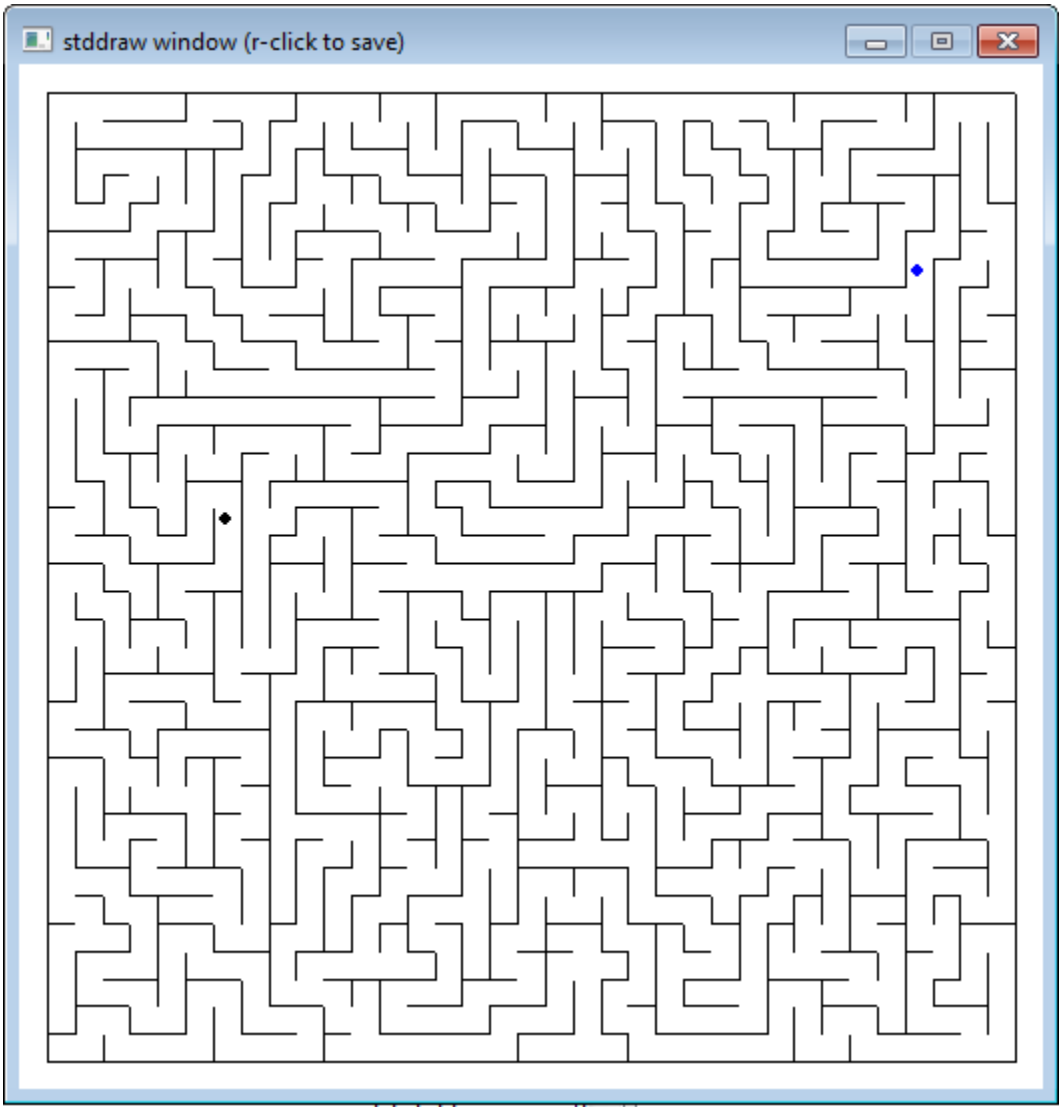
January 30th, 2023

Due Date: Sunday, 2/5/2023 11:59PM

Points: 35

Topic: Stacks, Queues, and Solving a Maze

In this assignment, you will be you will be using stacks and queues to solve a maze.



Data structures can help to solve problems, not just store data. Solving a maze is one such problem. Each time you visit a location in the maze, you add that position to your data structure and mark that you have visited that location. You keep exploring the maze in this manner until you run into a dead end (you are blocked by walls and the only way to get out is to retrace your steps, that is, move to spaces that have already been visited). When you run into a dead end, you remove a location from the data structure, according to its removal strategy, and set it as your current location.

For this assignment, you are provided with a class Maze.py that generates and draws a maze, and a class called Position that stores positions. You will need to modify the provided StackOfStrings.py code so that it works with Positions instead of Strings. You will also need to modify the provided QueueOfStrings.py code to do the same. You do not need to modify the Maze.py or Position.py, files. All the files you will need to complete the assignment are found in MazeLabStudent.zip

The Maze knows how to draw itself, and Positions also know how to draw themselves, so you will not write code to do the display. You will write code that calls the method to draw Positions, though. You will need to import StdDraw.py into your code and have it and its associated Python files in the same folder as your code so that the display works.

The main part of your assignment is to write a program called Solve.py that solves the maze in two ways: one using the stack data structure, and another using the queue data structure. Adding and removing from stacks and queues happens in a different order, so you should see different behavior for both solutions.

After following all instructions below, please submit your Python scripts (`Solve.py`, `StackOfPositions.py`, `QueueOfPositions.py`) and a writeup file (`Maze.docx`) in Moodle. Upload these as individual files – do not zip them. Be sure the script files have header sections that include the file name, your name, credits, a description of the file, the class and other useful information. Make sure your classes and all methods are thoroughly commented.

- `Solve.py`
- `StackOfPositions.py`
- `QueueOfPositions.py`
- `Maze.docx`

Grading:

This assignment is worth 35 points. You will be graded according to the following criteria:

Grade Item	Points
Program Compiles and Runs	2
Comments on all classes and methods	4
Revised StackOfPositions correctly	3
Revised QueueOfPositions correctly	3
Created and drew maze correctly	3
Solved with stack correctly	6
Solved with queue correctly	6
Outputs number of nodes visited for both solutions	2
Draws correctly during solutions	2
Writeup	5
Total	**30**

Getting Started

Download and Extract MazeLabStudent.zip

Create a folder and unzip the included `MazeLabStudent.zip` into that folder. This archive contains data files as well class files you will use. Note that in the assignment you need to create the `Solve.py` script and the `StackOfPositions.py and QueueOfPositions.py. The` rest of the scripts are already completed, and they should not be modified.

MazeLabStudent`.zip` includes the following files:

- Maze.py (defines the Maze class)
- Position.py (defines the Position class)
- QueueOfStrings.py (Queue example, use a starter for QueueOfPositions.py)
- StackOfStrings.py (Queue example, use a starter for StackOfPositions.py)
- color.py, picture.py, stdarray.py, stdio.py (helper files)
- StdDraw.py (defines drawing functions)

Part 1:

Modify the files StackOfStrings.py and QueueOfStrings.py so that they store and operate on Positions instead of Strings. Rename them StackOfPositions.py and QueueOfPositions.py. Don't forget to update the documentation to indicate the changes and add your name as an author. You should add comments for the class and all methods.

Part 2:

Write a program called Solve.py that creates a Maze using a command line argument for the dimension, n. Draw the maze, solve it using your StackOfPositions, drawing each new Position in StdDraw.RED, and each visited Position in StdDraw.BOOK_LIGHT_BLUE.

Then clear the maze (though not necessarily the drawing of the maze) by calling the appropriate Maze method and solve the maze using your QueueOfPositions. This time draw new Positions in StdDraw.DARK_RED and visited Positions in StdDraw.LIGHT_GRAY.

For each solving method, keep a count of how many nodes were added to the data structure in order to get to a solution, and print these out to the console.

Position Class and Maze Class:

The following two classes are provided for you. You do not need to alter the code in either of them, but you will need to instantiate and use them, via their methods.

Position Class (Position.py):

The Position class represents a location in the maze. It only knows its x and y position. The maze class keeps track of what has been visited or not

Here is the specification to use from the Position class. Do not modify the code.

```
Class Position
---------------------------

# create a position at maze coordinates x, y
__init__(int x, int y)

# returns the x position
int getX()

# returns the y position
int getY()

# draws the position as a circle of the specified color
void draw(Color color)

# return true if the Position is equal to the one passed in,
# false otherwise
boolean equals(Position p)

# returns a string representation of a Position
string toString()
```

Maze Class (Maze.py):

The Maze class represents the 2D grid of positions that make up a maze. For each position in the maze, it keeps track of whether there are walls to the north, east, south or west, and whether that position has already been visited.

Here is the specification to use from the Maze class. Do not modify the code.

```
class Maze
----------------------------

# create a maze with dimension nxn
__init__(int n)

# returns the start Position
Position getStart()

# returns the finish Position
Position getFinish()

# returns true if there is no wall to the north, false otherwise
boolean openNorth(Position p)

# returns true if there is no wall to the south, false otherwise
boolean openSouth(Position p)

# returns true if there is no wall to the east, false otherwise
boolean openEast(Position p)

# returns true if there is no wall to the west, false otherwise
boolean openWest(Position p)

# draws the maze
void draw()

# sets the visited flag for that Position in the maze to true
void setVisited(Position p)

# return true if the Position has been visited, false otherwise
boolean isVisited(Position p)

# clears all visited flags from the maze so another solution
# can be started
void clear()
```

Input and Testing

There is no input file. The size of the maze is provided as a command line argument. The maze is generated randomly. Test on small mazes first and then test on larger mazes to ensure your approach works.

Maze.docx (writeup)

Using no more than 1 page, write and submit a summary MS-Word document named Maze.docx. Include the following in this document:

Run your code multiple times and watch the behavior of the two approaches (stack vs queue). Submit a write up describing which one does better under what circumstances and why you think that is.

Helpful hints:

Use the example data files that are provided. Use the smaller ones to develop and test your code thoroughly before moving on to the larger data files.

Do not assume that because your code works on one data file that it will work on others. Test and debug thoroughly.

Work incrementally. Build and test incrementally, starting with easier aspects. Add functionality incrementally, testing with each change.

Review the examples from our book for text file input closely. Choose an approach that makes sense and works for you. There are different ways to accomplish the task at hand. Some are easier than others.

Review the slides and article about Defining Main Functions. Make sure you understand what is called for with the main function for this lab and how to make it work correctly in the several situations in which it will be used.

Review other related class slides.

No test script is provided for this lab. The mazes are generated at random and the solution changes with each maze.

There is no specification for the Solve program. You may define the methods and variables you need. Be sure to document them.

Do not modify the Maze.py or Position.py scripts.

Do not use Python's built in Stack and Queue classes. Use the provided code as specified, not Python library code.

Submit Your Work

After following all instructions above, please submit your Python script (Solve.py, StackOfPositions.py, QueueOfPositions.py) and a writeup file (Maze.docx) in Moodle. Upload these as individual files – do not zip them. Be sure the script files have header sections that include the file name, your name, credits, a description of the file, the class and other useful information. Make sure your class and all methods are thoroughly commented.

- `Solve.py`
- `StackOfPositions.py`
- `QueueOfPositions.py`
- `Maze.docx`

EXTENSIONS

You will not submit the answers to these extensions as part of your assignment submission. You should work through and understand them to learn more and practice. You may be held responsible for the content within, so you really should work through them. With that said, this are outside the scope of what will be submitted for this lab. Do not change your original submittal to address these extensions.

1. Research other "search" methods and implement one that is more intelligent on your own. Hint: You might google A* (pronounced A-star). Do not submit this.