**Maze Project**

**Please read this document in its ENTIRETY before starting on the project.**

# The Problem

A common challenge computer programmers are asked to solve is how to navigate a maze. For example, an object is placed at a certain position (the *starting* position) in the maze and is asked to try to reach another position (the *goal* position). Positions are defined by (*x*, *y*) coordinates and identified as *wall* or *open*.

At any given moment, the object can only move 1 step in one of 4 directions. Valid moves are:

* Up: (*x*, *y*) 🡪 (*x*, *y*-1)
* Right: (*x*, *y*) 🡪 (*x*+1, *y*)
* Down: (*x*, *y*) 🡪 (*x*, *y*+1)
* Left: (*x*, *y*) 🡪 (*x*-1, *y*)

The object can only move to open positions and be confined to the maze.

The challenge is to find a path from the starting position to the goal position (a *solution path*), if one exists.

# Objective

Write a program that represents a maze as an array, reads a maze specification, and writes out a graphic representation of the maze.

# Representation

The maze is graphically represented on the screen as follows:

A specification consists of:

* The size of the array is two integers: X, Y.
* The coordinate of the start square: (SX, SY).
* The coordinate of the finish square: (FX, FY).
* The coordinates of blocked squares: (BX1, BY1), (BX2, BY2), ... ,(BXn, BYn).

The graphical output consists of 3x3 squares of one of the following forms (using a fixed-width font)

...

.S. for a starting location

...

...

.F. for a finishing location (*goal*)

...

XXX

XXX for a blocked square (*wall*)

XXX

...

... for all other squares

...

and is annotated by coordinates.

For example:

Your program should read the maze data from a file called *input\_file* and output the results into a file called *output\_file*.

If *input\_file* contained:

10, 11.

(0, 0).

(3, 10).

(0, 6), (0, 7), (0, 9),

(1, 1), (1, 2), (1, 4), (1, 7),

(2, 1), (2, 7), (2, 8), (2, 10),

(3, 1), (3, 2), (3, 3), (3, 4), (3, 7),

(4, 0), (4, 6), (4, 9), (4, 10),

(5, 2), (5, 3), (5, 4), (5, 5), (5, 7),

(6, 1), (6, 8), (6, 9),

(7, 1), (7, 2), (7, 3), (7, 6),

(8, 1), (8, 5), (8, 6), (8, 8), (8, 10),

(9, 3), (9, 7).

(notice the use of the . as a sentinel.)

Then, the program should produce *output\_file* with:

0 1 2 3 4 5 6 7 8 9

............XXX...............

0 .S..........XXX...............

............XXX...............

...XXXXXXXXX......XXXXXXXXX...

1 ...XXXXXXXXX......XXXXXXXXX...

...XXXXXXXXX......XXXXXXXXX...

...XXX...XXX...XXX...XXX......

2 ...XXX...XXX...XXX...XXX......

...XXX...XXX...XXX...XXX......

.........XXX...XXX...XXX...XXX

3 .........XXX...XXX...XXX...XXX

.........XXX...XXX...XXX...XXX

...XXX...XXX...XXX............

4 ...XXX...XXX...XXX............

...XXX...XXX...XXX............

...............XXX......XXX...

5 ...............XXX......XXX...

...............XXX......XXX...

XXX.........XXX......XXXXXX...

6 XXX.........XXX......XXXXXX...

XXX.........XXX......XXXXXX...

XXXXXXXXXXXX...XXX.........XXX

7 XXXXXXXXXXXX...XXX.........XXX

XXXXXXXXXXXX...XXX.........XXX

......XXX.........XXX...XXX...

8 ......XXX.........XXX...XXX...

......XXX.........XXX...XXX...

XXX.........XXX...XXX.........

9 XXX.........XXX...XXX.........

XXX.........XXX...XXX.........

......XXX...XXX.........XXX...

10......XXX.F.XXX.........XXX...

......XXX...XXX.........XXX...

# Project Descriptions

This maze project has seven parts throughout the course. Each part is built upon or is an extension of previous parts. You may have to modify (*refactor*) previous sections to proceed to the next parts. Therefore, it is very important that your code is clean as the parts become more complicated.

**Project: Maze 1 (Due at the end of Topic 1)**

Write a program that reads in a description of a 2-dimensional maze with a Start and a Goal and then displays the maze. Use a two dimensional array of objects representing the squares. (This program will be extended and “refactored” several times during the class, so it is important to separate concerns into distinct single-purpose objects with minimal coupling.)

**Project: Maze 2 (Due at the end of Topic 2)**

Replace the 2-dimensional array of objects in your *Maze 1* program with linked lists of the same objects. The linked lists can be single, double, or circular as you deem appropriate for your implementation. Make as few changes as possible in the rest of the program. Write up what changes to *Maze 1* would have reduced the number of such changes necessary. In retrospect, how would you have written *Maze 1* differently?

**Project: Maze 3 (Due at the end of Topic 3)**

Use a stack to find paths through the maze (from Start to Goal) utilizing your maze program. Illustrate the progress through the maze at each step. Your program should visually distinguish between the path identified and any backtracking that may have occurred. In addition, illustrate the content of the stack at each step.

**Project: Maze 4 (Due at the end of Topic 3)**

Use a queue to find paths through the maze (from Start to Goal) utilizing your maze program. Illustrate the progress through the maze at each step. Your program should visually distinguish between the path identified and any backtracking that may have occurred. In addition, illustrate the content of the queue at each step.

**Project: Maze 5 (Due at the end of Topic 5)**

Implement a heap (as a priority queue where priority is defined as the Manhattan distance from the goal). Use the heap to find paths through the maze (from Start to Goal) utilizing your maze program. Your program should visually distinguish between the path identified and any backtracking that may have occurred. In addition, illustrate the content of the heap at each step.

**Project: Maze 6 (Due at the end of Topic 7)**

Modify your *Maze 3* to represent the maze as a graph and solve by Depth First Search.

**Project: Maze 7 (Due at the end of Topic 7)**

Modify your *Maze 4* to represent the maze as a graph and solve by Breadth First Search.

# Deliverables

You are expected to follow these standard guidelines for all parts.

### Implementation notes:

1. Write your code in the language specified by your instructor.
2. The input to your program should be screen based or from a file, as directed by your instructor.
3. The output of your program should appear on screen and/or in a file, as directed by your instructor.
4. There are always multiple ways to code a project. In class, we will discuss several options. It is up to you to choose the implementation you prefer. Your code must perform all the required tasks and all other requirements in this document must be met.
5. Your code must be fully commented, but focus on the main programming blocks.
6. Include a screenshot of the running program.
7. When an error in reading data is detected, your program should alert the user in a helpful manner (think of it as customer service to the user).

### Submit the following:

1. **By Day 5 of the week assigned:** a summary of the main purpose of your project, a detailed description of your implementation approach, and a code skeleton (i.e., class with stubs for all methods)
2. **By the assignment due date:** complete running program, commented code, the input file, and the entire code project as one zipped file. Include a ReadMe file with instruction on how to run it and what software is required.

This assignment uses a grading rubric. Instructors will be using the rubric to grade the assignment; therefore, students should review the rubric prior to beginning the assignment to become familiar with the assignment criteria and expectations for successful completion of the assignment.

You are not required to submit this assignment to Turnitin.