Project 01: Maze Explorer COSC 102: Spring 2019

1 Maze Explorer: Part I - Representing Mazes

1.1 Learning Outcomes

This homework is an application of procedural programming, the programming paradigm learned in COSC101. This first implementation puts into practice your readings of the textbook (BJP) and the design concepts we are emphasizing during lectures and labs. Read this entire document before starting your work.

In this first Java programming assignment you will practice:

1. using objects (such as `String` and `Scanner`),
2. creating and traversing Java's most basic data structure, arrays,
3. dealing with input from files and from standard input, i.e., the console and generating output, i.e. text on the console; input/output also take the form of function parameters and return values,
4. writing programs using top down design and
5. writing functions that are designed according to the SOFA properties:
   - Short - they execute a short function body,
   - One - they do only one thing
   - Few - they take few parameters (two or three is good), and
   - Abstraction - they are based on an abstraction

   and DNY (don't repeat yourself).

Specifically, in Java you will implement code to:

- represent a maze based on an input file,
- generate its ASCII "art" output on the file console and
- explore the maze's content according to a location entered by the user at the console

In lab this week, you will be assigned your partner for this project. You will work with your partner, using the pair programming development method, an efficient and collaborative practice commonly used in industry. You will demo the functionality for this assignment in next week's lab period. At which point you will be given feedback to consider in revising your program, as well as Part II of the Maze Explorer programming assignment. Part II will be completed individually.

1.2 Maze Construction

Description Your program will use information from an input file to define a maze and display an ASCII version of that maze on the console.

The following describes the format of the maze file:

- The first line of the file represents the size of the maze. It will contain two numbers:

  - The first number represents the height of the maze.
  - The second number represents the width of the maze.

  Note: There is no location 0 0, the top left corner of the maze is considered to be location 1 1.

- Subsequent lines of the file represent elements found within the maze boundaries and their locations. There are two types of elements:

  1. treasures
  2. obstacles

  An element is represented by the first character in the line. The first character will be followed by either two or four numbers, that specify the element's position in the maze:

  - Two numbers: The element is a treasure. The first number is the row, the second number is the column.
  - Four numbers: The element is an obstacle. The first two numbers specify the row and column, respectively, of the upper-left corner of the obstacle. The last two numbers specify the row and column of the bottom-right corner.

  Note: You may assume that the writer of the input file is responsible for ensuring all elements are fully contained within the maze boundaries and that no maze elements overlap.

For example, given the file `plan1.txt`, shown below, the maze contains one treasure, '$', and one obstacle, '*':

```
3 5
$ 1 2
* 3 4 3 4
```

the ASCII output displayed in the console should be:

```
+-----+
|.$...|
|.....|
|...*.|
+-----+
```

Note: A '.' inside the maze represents an empty space. The '+', '-', '|' characters form the border of the maze, the active parts of the maze are within the border.

Hint: If you have difficulty reading the file initially, you can make progress by starting with a hard-coded version.

1.3 Program Review I

Before continuing to section 1.4, check your program's design:

- draw the hierarchal diagram of your program
- answer the following questions in the provided README file:

  1. Does your program represent data in an easy to use and organized manner? Explain.
  2. Do your choices to represent data as class variables, constants or passed parameters make sense? Explain.
  3. Does each function in your program meet SOFA requirements? Describe which aspects of your functions could be improved.
  4. It is likely you have lines of code that are almost repeats in your program. Explain how you can abstract them to a function to avoid redundancy. Describe where you refactored your code to reduce repetition.

5. How many helper methods do you have? Are there functions that look the same, i.e., similar parameters and bodies, or serve the same or similar purpose? If you have less than 4, you don't have enough. If you have more than 6, you may have too many. Describe where you refactored your functions.
6. Does each of your methods have a description?

- revise and improve your program based on what you have discovered by answering the above questions

Once you have answered the questions and improved your code, continue to the next section.

1.4 Cell Queries

Description After loading a maze, the program should repeatedly query the user for a cell coordinate input: two integers, row first and column second. According to the cell entered, your program provides feedback: the cell is either outside the maze, a treasure, within an obstacle, or empty. Entering 0 0 terminates the repetition.

For example, using the provided file `plan2.txt`, the output of the program (with the shown user responses) would look like the following:

```
Please enter the grid file name: plan2.txt

+------------+
|.........*..|
|.***....$*..|
|.***.....*..|
|.........*..|
|....*.......|
|............|
|......******|
|$.....******|
+------------+

Enter a cell (row and column; zeros to exit): 10 8
(10, 8) is out of the maze
Enter a cell (row and column; zeros to exit): 2 9
(2, 9) is a treasure
Enter a cell (row and column; zeros to exit): 3 3
(3, 3) is within an obstacle
Enter a cell (row and column; zeros to exit): 4 3
(4, 3) is an empty cell
Enter a cell (row and column; zeros to exit): 0 0
Goodbye
```

1.5 Program Review II

At this point you should again review your program according to the questions in the Program Review I section above. You are not required to answer the questions in the README file this time. Simply review the program again and make your improvements.

Update your hierarchal diagram and turn in only the final version.

1.6 Submission and Demo

1.6.1 Submission

Your project Part I must be submitted to your lecture section's Moodle page by the beginning of your lab time period, during the week of February 4th.

Your Moodle submission should include:

1. Your .java file (do not submit the .class file)
2. A diagram of the hierarchical structure of your program in .pdf format. (There are many programs available for drawing diagrams, for example, Google Drawings that can then be downloaded as a .pdf to submit).
3. A plain text file that specifies a maze of your own creation that you used for testing.
4. README.txt with questions answered.

Submit to your lecture instructor:

1. Your pair programming log.
2. Your pair programming assessment. (completed individually)

1.6.2 Demo

On the due date, you will demo your project to your lab instructor. Please come prepared to Demo your working program and bring a print out of the hierarchical structure of your program to hand in.

1.7 Grading

Part I will be graded based on the following criteria:

1. Correctness: 15%
2. Program design and style: 10%
3. Peer Programming and Design Documents 5%

Each of the above criteria will be given a check, check plus, or check minus as well as feedback on how to improve your program.

2 Maze Explorer - Part II - Exploring Mazes

2.1 Learning Outcomes

In this part of the assignment you will continue practice of Java procedural programming and program design concepts, this time **individually**.

Starting with the code your wrote with your partner for part I, you will add:

1. labels for the maze display allowing easier player interaction,
2. maze navigation via a player entered path, with program feedback and
3. a game aspect with the goal of collecting all the treasures.

2.2 Augmenting the Maze Display

Description The first feature to develop individually is to label the rows and columns of the maze display on the console so that the player has an easier time navigating the maze. When the row or column is more than one digit, the last digit only should be used.
For example, executing the program and entering `plan3.txt` should display the output below on the console.

```
Please enter the grid file name: plan3.txt

  12345678901234
 +--------------+
1|....*.....*...|
2|........$.*...|
3|...***....*.Y.|
4|$..***........|
5|.......**.....|
 +--------------+
```

Note: Treasures are defined by any character followed by two integers.

2.3 Maze Explorer Navigation

Description Instead of a single cell query, this version of the program asks the player for a starting cell within the maze and asks the player for a path moving through the maze. The program will then provide feedback on the explorer's final location based on the entered moves.

Moves are indicated by the following characters:

- '>' move right,
- '<' move left,
- '^' move up,
- 'v' move down (lowercase only)

For example, continuing the example in 2.2, after displaying the labeled maze on the console, the following represents a sequence of program queries, user input and program responses:

```
Enter the start cell (row and column; zeros to exit) and the moves: 1 3 >v>>>
Explorer ends at (2, 7)

Enter the start cell (row and column; zeros to exit) and the moves: 3 5 >v>^^
Explorer can't start within an obstacle...

Enter the start cell (row and column; zeros to exit) and the moves: 5 6 <^<<<
Explorer crashed into an obstacle at (4, 5)...

Enter the start cell (row and column; zeros to exit) and the moves: 4 2 <<<<
Explorer left the grid at (4, 0)...

Enter the start a cell (row and column; zeros to exit) and the moves: 0 0
Goodbye
```

2.4 Program Review III

Once again, before continuing, you should review your program according to the questions in the Program Review I section above. Document any part of your program you identify that needs refactoring and state why in the README file for part II. Make improvements based on your findings.

2.5 Collecting Treasure

Description Now we are ready to turn this explorer program into a game. The program will provide the player with a random valid starting position in the maze and the player's goal is to collect all the treasures. From the starting position, the player must enter a path that visits all locations with a treasure, thereby 'collecting' them and winning the game.

The program evaluates the path the player enters with the following possible outcomes:

1. The game reports a win and exits the program: all the treasures are collected with a path that did not leave the grid or crash into an obstacle at any point.
2. The game resets the treasure(s) and prompts the player to try again from the original starting position if the path:
    - causes the explorer to leave the grid,
    - crashes into an obstacle, or
    - fails to collect all the treasures

Note: If the player collects all the treasures, but then crashes into an obstacle or leaves the grid the game does not report a win.

```
Please enter the grid file name: plan2.txt

  123456789012
 +------------+
1|.........*..|
2|.***....$*..|
3|.***.....*..|
4|.........*..|
5|....*.......|
6|............|
7|......******|
8|$.....******|
 +------------+

Explorer starting point is (7, 1).
Enter moves to collect all the treasure. (zeros to exit): v>^^^^^>>>>>>>^
Explorer crashed into an obstacle at (3, 2)... try again!

Explorer starting point is (7, 1).
Enter moves to collect all the treasure. (zeros to exit): v>^^^^>>>>>>>^^
Explorer collected all the treasures!
You won the game!!
```

2.6 Program Review IV

Do a final review ensuring your program meets all design criteria. Review your program according to the questions in the Program Review I section above. Document any part of your program you identify that needs refactoring and state

why in the README file for part II . Make improvements based on your findings.

Update your hierarchal diagram to include all the functionality added in part II.

2.7 Extra Credit

Before you begin this optional part of the assignment be sure you have a copy of the program that matches the description above saved without any extra credit modifications. Save a new copy of your program with `ExtraCredit` in the file and class name and make your changes there. Hand in both the original and the extra credit file as part of your submission.

Description There are many enhancements that could be added to make this game a little more interesting. Be creative and think of your own!

Any additions should be well designed and documented in your `README.txt` and labeled 'Extra Credit'. Make sure to describe how the player should interact with the changes.

2.8 Submission

Your project Part II must be submitted to your lecture section's Moodle page by 10:00pm on Friday, February 15, 2019.

Your Moodle submission should include:

1. Your .java file (do not submit the .class file).
2. A diagram of the hierarchical structure of your program in .pdf format. (There are many programs available for drawing diagrams, for example, Google Drawings that can then be downloaded as a .pdf to submit).
3. README.txt documenting program changes you made to improve design.
4. (optional) extra credit .java file.

2.9 Grading

Part II will be worth 70% of your grade. It will be graded based on the following criteria:

1. Correctness: 40%
2. Program design and style: 20%
3. Design Documents 10%