Simon Fraser University
School of Computing Science

Cmpt 120 – Assignment 2

A Simple Game of Mario

Objective

Our objective, in this Assignment 2, is to …

- File I/O: Practice reading data from a text file,
- Software Development using Python: Translate a "high-level" algorithm into corresponding Python statements, and
- Have fun along the way.

Before Doing This Assignment

I strongly recommend you do the Practice Exercises of Lecture 31 on Friday July 19.

Problem Statement

In Assignment 2, you are asked to **complete** a simple game of Mario, given the partially implemented Python program `My Mario Game.py`.

How the game works

The goal of this game is for the player to move Mario through a maze (two-dimensional grid displayed on the screen) one cell at a time (either right, left, up or down) such that Mario reaches the exit of the maze with a score greater than 0.
This is not as easy as it sounds since buried in the maze are bombs which will reduce Mario's score by 1 every time Mario lands on such cell. On the hand, Mario's score is incremented by 1 every time Mario lands on a cell containing a treasure. There are also empty cells in the maze, i.e., cells that do not contain any obstacles and therefore do not affect Mario's score.

The game ends (and the player loses) as soon as Mario's score is reduced to 0. The player wins when Mario passes through the exit gate of the maze with a score greater than 0.

Sample Runs

To illustrate how the game is played, we have created three Sample Runs, each illustrating a particular scenario: the player loses the game, the player exits the game, the player wins the game. These three scenarios can be found in the following files, respectively:

- `Assn 2 Sample Run 1.txt`
- `Assn 2 Sample Run 2.txt`
- `Assn 2 Sample Run 3.txt`

Go over the content of these files carefully as they will help you understand how the game works.

How to complete the game

Have a look at the given Python program `My Mario Game.py`. You are asked to complete two sections of this program: Part 1 and Part 2.

Part 1

You need to complete the function `readDataFileAndSetVariables( filename )` in which you are to read the data found in the given data file `Assn 2 InputData.txt` and assign this data to several variables.

Format of `Assn 2 InputData.txt`

In order to complete this Part 1 and write Python code that properly read the data contained in the input data file `Assn 2 InputData.txt`, you need to know how the data contained in this file is structured. So, let's have a look at its format:

- Line 1 is the **width** of the maze.
- Line 2 is the **height** of the maze.
- Line 3 is the **number of treasures** (to be buried in the maze).
- Line 4 is the **number of bombs** (to be buried in the maze).
- Line 5 is the **symbol representing an empty maze cell**. Note: there is a blank space before the symbol then a blank space after it.
- Line 6 is the **symbol representing a treasure** contained in a maze cell. Note: there is a blank space before the symbol then a blank space after it.
- Line 7 is the **symbol representing a bomb** contained in a maze cell. Note: there is a blank space before the symbol then a blank space after it.

- Line 8 is the **symbol representing Mario** contained in a maze cell. Note: there is a blank space before the symbol then a blank space after it.
- Line 9 is the **symbol representing the exit gate located on either the top or bottom border**. Note: there is a blank space before the symbol then a blank space after it.

Note: The reason there is a blank space before each of the above 5 symbols then a blank space after each of them is because each cell of the maze is 3-character wide.

- Line 10 is the **symbol used to create the top and bottom horizontal borders**. This is the border drawn at the top and bottom of the maze when the maze is drawn on the computer monitor screen. Note that these symbols are not contained in the maze.

Note: The reason the above symbol is composed of 3 hyphens ('-') is because each cell of the maze is 3-character wide.

- Line 11 is the **symbol used to create the left and right vertical borders**. This is the border drawn to the left and right of the maze when the maze is drawn on the computer monitor screen. Note that these symbols are not contained in the maze.
- Line 12 is the **location** (row and column) **of Mario**, i.e., coordinates of Mario in the maze at the start of the game. Careful! The row is a number between 0 and mazeHeight-1 and the column is a number between 0 and mazeWidth-1.
- Line 13 to Line 27 (inclusively) are the locations of the treasures, one location (row and column) per line. Remember, there are a certain number of treasures and this number was found on Line 3 of this file. Careful! The row is a number between 0 and mazeHeight-1 and the column is a number between 0 and mazeWidth-1.
- Line 28 to Line 57 (inclusively) are the locations of the bombs, one location (row and column) per line. Remember, there are a certain number of bombs and this number was found on Line 4 of this file. Careful! The row is a number between 0 and mazeHeight-1 and the column is a number between 0 and mazeWidth-1.
- Line 58 is the **bomb to score ratio**, used to set the initial value of Mario's score.

You may wonder: Why putting all this data into a file, which you then open, read its data one line at a line and assign this data to corresponding variables? Why not assigning the data directly to corresponding variables (i.e., *hard-coding* the data) in the program?

Here is an example of what we mean by *hard-coding* the data in the program:

```
mazeWidth = 10
mazeHeight = 10
mario = 'M'
```

The reason why we put all the data into an input file is that it allows us to customize our Simple Game of Mario, for example, changing the dimensions of its maze, the numbers of bombs and/or treasures buried in its maze, the symbol used to represent Mario, etc… easily **without having to modify the program**. Remember, modifying code is always error-prone, i.e., an opportunity to introduce errors in a program. So, once a program executes as expected, software developers try real hard not to have to modify it. And reading data from an input file is one way to achieve this.

Part 2

You need to complete the main part of the program. Again, have a look at the given Python program `My Mario Game.py`.

In the main part of this program, you will find a “high-level” algorithm describing the part you need to complete. This algorithm describes the “engine” behind the game, i.e., the code that makes Mario move and that makes the game progress.

By the way, “high-level" means "not very detailed". So, your task is to discover these details: what does your code need to do in order to move Mario one cell either right, left, up or down the maze (depending on the input from the player) and to make the game progress. Once these details are clear, document them as comments (add more comments) in the program then translate them into Python statements.

Your program is completed and ready to be submitted when it produces **exactly** what is displayed in the three given sample runs, when the player enters the same input, without producing errors or crashing.

Requirements

When completing this assignment, you cannot modify the code provided in given Python program `My Mario Game.py`. You simply need to add your own code to the provided code.

Easiest Way to Proceed with this Assignment

Below are suggestions that may make your life easier when completing this assignment:

1. First, go over the given code and understand what it does.

2. Do Part 1. Once you have completed the function `readDataFileAndSetVariables( filename )`, test it using the code (several `print` statements) labelled `# For debugging purposes`. These `print` statements have been commented out, so you will need to uncomment them first before executing your program. These `print` statements will allow you to verify that you have indeed read the correct values from the input data file. Once you are satisfied with the results, you will need to comment them out again.

3. Move on to Part 2. The first thing you may wish to do is uncomment the Python statement:

   ```
   #displayMaze(theMaze, mazeWidth, mazeHeight, hBoundary,
   boundarySide)
   ```

found under

```
# Display the maze
# Once you have completed Part 1, uncomment the following
Python statement and see what is displayed on the screen!
#displayMaze(theMaze, mazeWidth, mazeHeight, hBoundary,
boundarySide)
```

This statement displays the completely constructed maze on the computer monitor screen. Again, this will allow you to verify that the code is working as expected up to this point.

4. Once this is done, start incrementally developing your code using the given algorithm. You may wish to implement one comment at a time.

5. Feel free to uncomment and make use of any statement labelled `# For debugging purposes` while you are testing your code.

6. While testing your code, you can set the data stored on Line 6 of `Assn 2 InputData.txt`, which is the symbol representing a treasure contained in a maze cell, to another character. This will allow you to differentiate between a treasure and a bomb (in the maze) and will ease your testing/debugging.

7. As you are working on Part 2 of this assignment, try to make use of the code (functions) already provided in `My Mario Game.py` as much as you can. For example, can you make use of the function `placeInMaze(…)`?

Submission

- Submit the completed file **`My_Mario_Game.py`** on CourSys by Thursday, August 1, 2019, no later than 15:00.
- You can submit your work to CourSys as many times as you wish, but it is the last submission that counts. So, let's make sure that your last submission is done before the deadline.
- No late submission will be accepted: any submission made after the due date and time stated above will not be marked for grades. However, late submissions will be marked to provide feedback to students.
    - The only exception is for medical reasons. If you are not able to submit your assignment on time due to a medical reason, you must notify the instructor by email right away and provide her with a doctor's note as soon as possible. Use the doctor's note form found on the course web site.

Marking

When marking Assignment 2, we will look at some of the following criteria:

(Note: In the criteria below, when we say "your program", we often mean "the parts you added to the given program".)

- Whether your program executes: are there any syntax errors.
- How well your program's behaviour matches the description in the game given in this assignment and illustrated in the three Sample Run files provided.
- How well your program abides to the Good Programming Style described on the GPS web page of the course web site.
- Whether all the requirements of this assignment are satisfied.
- Whether all your functions are located at the appropriate place in your program.
- Whether functions are called either from the "# Main" part of your program and/or from other functions and whether all functions are called at least once.
- Whether your code is easy to read and well commented.
- Whether you selected the most appropriate/efficient conditional and iterative statements for each situation in your program.
- Whether you are the only author of the code you have added to the provided program.
- Whether your Python program is a real program and no simply a screenshot of the Python Interpreter Shell window onto which you have entered some Python statements.

Note: When we test your program, we will be using `Assn 2 InputData.txt`.

Have fun!

Anne Lavergne – July 2019