Scripting Languages: Project 3

December 1, 2020

Project Guidelines

- This project is an individual project.
- The program output should have exactly the same format as the provided sample output.
- If a specific aspect of the project is not working, you will still get a partial grade on the implemented parts of the project. In that case, it may be useful to add some comments on the problem and on the steps you tried to solve it.
- The project is corrected on 10 points. The bonus task 11 counts for 1 extra point.

The assignment

This project is based on project 2. In this project, you will create a fully functional version of the terminal-based Minesweeper by adding the following features.

1. When the user chooses to uncover a cell with no neighboring bombs (i.e. a dot `.`), uncover all neighboring cells until reaching a number.
2. The player can label a cell as a bomb and track the number of bombs during the game.
3. The player can store a game to a file, and can load a game file to continue playing.

These features should be implemented by using Object-Oriented Programming (OOP).

An example game play looks as follows. The player's input is marked blue. When a game starts, all cells are covered (marked as `#`) and the number of marked bombs is printed. The player should enter one string `option` and two integers `row` and `column` to indicate which cell he or she wishes to uncover or mark as a bomb.

```
    1 2 3 4 5 6 7 8
   ----------------
1 | # # # # # # # #
2 | # # # # # # # #
3 | # # # # # # # #
4 | # # # # # # # #
5 | # # # # # # # #
6 | # # # # # # # #
7 | # # # # # # # #
8 | # # # # # # # #
Number of marked bombs: 0
(a) uncover a cell
(b) mark or unmark a bomb
Please enter your option: a
Uncover the cell at row: 6
                 column: 2

    1 2 3 4 5 6 7 8
   ----------------
1 | # # # # # # # #
2 | # # # # # # # #
3 | # # # # # # # #
4 | 1 2 # # # # # #
5 | . 1 2 # # # # #
6 | . . 1 # # # # #
7 | 1 1 1 # # # # #
8 | # # # # # # # #
Number of marked bombs: 0
(a) uncover a cell
(b) mark or unmark a bomb
Please enter your option: a
Uncover the cell at row: 2
                 column: 5

    1 2 3 4 5 6 7 8
   ----------------
1 | # # 1 . . 1 # #
2 | # # 2 . . 1 # #
3 | # # 3 1 . 1 1 1
4 | 1 2 # 2 1 . . .
5 | . 1 2 # 1 . . .
6 | . . 1 # 2 2 3 2
7 | 1 1 1 # # # # #
8 | # # # # # # # #
Number of marked bombs: 0
(a) uncover a cell
```

```
(b) mark or unmark a bomb
Please enter your option: a
Uncover the cell at row: 3
                column: 1

    1 2 3 4 5 6 7 8
   ----------------
1 | # # 1 . . 1 # #
2 | # # 2 . . 1 # #
3 | 2 # 3 1 . 1 1 1
4 | 1 2 # 2 1 . . .
5 | . 1 2 # 1 . . .
6 | . . 1 # 2 2 3 2
7 | 1 1 1 # # # # #
8 | # # # # # # # #
Number of marked bombs: 0
(a) uncover a cell
(b) mark or unmark a bomb
Please enter your option: b
Mark or unmark a bomb at row:3
                     column:2

    1 2 3 4 5 6 7 8
   ----------------
1 | # # 1 . . 1 # #
2 | # # 2 . . 1 # #
3 | 2 @ 3 1 . 1 1 1
4 | 1 2 # 2 1 . . .
5 | . 1 2 # 1 . . .
6 | . . 1 # 2 2 3 2
7 | 1 1 1 # # # # #
8 | # # # # # # # #
Number of marked bombs: 1
(a) uncover a cell
(b) mark or unmark a bomb
Please enter your option: b
Mark or unmark a bomb at row:4
                     column:3

    1 2 3 4 5 6 7 8
   ----------------
1 | # # 1 . . 1 # #
2 | # # 2 . . 1 # #
3 | 2 @ 3 1 . 1 1 1
4 | 1 2 @ 2 1 . . .
5 | . 1 2 # 1 . . .
```

```
6 | . . 1 # 2 2 3 2
7 | 1 1 1 # # # # #
8 | # # # # # # # #
Number of marked bombs: 2
(a) uncover a cell
(b) mark or unmark a bomb
Please enter your option: a
Uncover the cell at row: 1
                column: 7

    1 2 3 4 5 6 7 8
   ---------------
1 | # # 1 . . 1 2 #
2 | # # 2 . . 1 # #
3 | 2 @ 3 1 . 1 1 1
4 | 1 2 @ 2 1 . . .
5 | . 1 2 # 1 . . .
6 | . . 1 # 2 2 3 2
7 | 1 1 1 # # # # #
8 | # # # # # # # #
Number of marked bombs: 2
(a) uncover a cell
(b) mark or unmark a bomb
Please enter your option: b
Uncover the cell at row: 4
                column: 3


    1 2 3 4 5 6 7 8
   ---------------
1 | # # 1 . . 1 2 #
2 | # # 2 . . 1 # #
3 | 2 @ 3 1 . 1 1 1
4 | 1 2 # 2 1 . . .
5 | . 1 2 # 1 . . .
6 | . . 1 # 2 2 3 2
7 | 1 1 1 # # # # #
8 | # # # # # # # #
Number of marked bombs: 1
(a) uncover a cell
(b) mark or unmark a bomb
Please enter your option: a
Uncover the cell at row: 5
                column: 4

    1 2 3 4 5 6 7 8
```

```
   ---------------
1 | # # 1 . . 1 2 *
2 | # * 2 . . 1 * #
3 | 2 @ 3 1 . 1 1 1
4 | 1 2 * 2 1 . . .
5 | . 1 2 * 1 . . .
6 | . . 1 # 2 2 3 2
7 | 1 1 1 # # * * *
8 | # * # # # # # #
Number of marked bombs: 1
Oh no! You lose!
Thank you for playing Minesweeper!
```

Task 1 - `new_game()`

Implement a function `new_game()` that returns a new map that has 10 randomly placed bombs. A map is a list of lists (for the rows and columns) that stores the cells. A cell is an object of the `Cell` class that has the following attributes:

- `sign`: A character in `["1", "2", "3", "4", "5", "6", "7", "8", ".", "*"]`
- `uncovered`: A Boolean value indicating whether the cell is uncovered
- `marked`: A Boolean value indicating whether the cell is marked as a bomb
- `map`: The map the cell belongs to
- `neighbors`: A list of neighboring `Cell` objects.

Each cell in the returned map should have the following settings. Its `map` is initially set to the map it belongs to. Its `sign` is initially set to `.` (a dot). Its `uncovered` and `marked` are initially set to `False`. Its `neighbors` is initially set to a list of neighboring cells.

Hint: You can adapt functions such as `set_bombs()`, `neighbors(row, column)`, `new_game()` in project 2 to this new cell representation.

Task 2 - `mark_unmark()`

Implement a `mark_unmark()` function in the `Cell` class. This function flips the cell's `marked` value. That is, for a cell having `marked` as `True`, this function sets it to `False`, and vice versa.

Task 3 - `print_map(map)`

Implement a function `print_map(map)` that prints a given map in the required format. Each cell in the given map is a `Cell` object. If the cell is marked as a bomb, a `@` is printed. If the cell is not marked and is uncovered, its sign is printed. If the cell is not marked as a bomb and is covered, a `#` is printed. This function also tracks the number of marked bombs. When `print_map(map)` is called, the effect should be that we see the following on the screen.

Example 1

```
    1 2 3 4 5 6 7 8
   ----------------
1 | # # 1 . . 1 2 #
2 | # # 2 . . 1 # #
3 | 2 @ 3 1 . 1 1 1
4 | 1 2 @ 2 1 . . .
5 | . 1 2 # 1 . . .
6 | . . 1 # 2 2 3 2
7 | 1 1 1 # # # # #
8 | # # # # # # # #
Number of marked bombs: 2
```

Example 2

```
    1 2 3 4 5 6 7 8
   ----------------
1 | # @ 1 . . 1 2 #
2 | # @ 2 . . 1 # #
3 | 2 @ 3 1 . 1 1 1
4 | 1 2 @ 2 1 . . .
5 | . 1 2 # 1 . . .
6 | . . 1 # 2 2 3 2
7 | 1 1 1 # # # # #
8 | # # # # # # # #
Number of marked bombs: 4
```

Example 3

```
    1 2 3 4 5 6 7 8
   ----------------
1 | # # 1 . . 1 2 *
2 | # * 2 . . 1 * #
3 | 2 @ 3 1 . 1 1 1
4 | 1 2 * 2 1 . . .
5 | . 1 2 * 1 . . .
6 | . . 1 # 2 2 3 2
```

```
7 | 1 1 1 # # * * *
8 | # * # # # # # #
Number of marked bombs: 1
```

Task 4 - `get_input(map)`

Implement a function `get_input(map)` that takes a map and returns a list of one string and two integers `[option, row, column]`. The returned `option` indicates the player's next move. The returned integers `row` and `column` indicate a location on the map. The function keeps asking until the player enters a valid option `a` or `b`. And after that, the function keeps asking until the player provides a valid location. The player is not allowed to choose `a` if all cells are either uncovered or marked as a bomb (i.e. if there is no `#` in the map to be uncovered). The player is not allowed to mark or unmark a cell that is already uncovered.

We give three examples.

Example 1

```
    1 2 3 4 5 6 7 8
   ----------------
1 | # # 1 . . 1 2 #
2 | # # 2 . . 1 # #
3 | 2 @ 3 1 . 1 1 1
4 | 1 2 @ 2 1 . . .
5 | . 1 2 # 1 . . .
6 | . . 1 # 2 2 3 2
7 | 1 1 1 # # # # #
8 | # # # # # # # #
```

With the map above, `get_input(map)` should behave as follows.

```
(a) uncover a cell
(b) mark or unmark a bomb
Please enter your option: 1
Invalid option. Please try again.
(a) uncover a cell
(b) mark or unmark a bomb
Please enter your option: b
Mark or unmark a bomb at row: 4
                      column: 1
The cell was already uncovered. Please try again.
Mark or unmark a bomb at row: 3
                      column: 2
```

And a list `["b", 3, 2]` is returned.

Example 2

```
   1 2 3 4 5 6 7 8
  ----------------
1 | # # 1 . . 1 2 #
2 | # # 2 . . 1 # #
3 | 2 @ 3 1 . 1 1 1
4 | 1 2 @ 2 1 . . .
5 | . 1 2 # 1 . . .
6 | . . 1 # 2 2 3 2
7 | 1 1 1 # # # # #
8 | # # # # # # # #
```

With the map above, `get_input(map)` should behave as follows.

```
(a) uncover a cell
(b) Mark or unmark a bomb
Please enter your option: a
Uncover the cell at row: one
                 column: one
Invalid row or column. Please try again.
Uncover the cell at row: 3
                 column: 2
The cell is marked as a bomb. Please try again.
Uncover the cell at row: 1
                 column: 1
```

And a list `["a", 1, 1]` is returned.

Example 3

```
   1 2 3 4 5 6 7 8
  ----------------
1 | @ @ 1 . . 1 2 @
2 | @ @ 2 . . 1 @ @
3 | 2 @ 3 1 . 1 1 1
4 | 1 2 @ 2 1 . . .
5 | . 1 2 @ 1 . . .
6 | . . 1 @ 2 2 3 2
7 | 1 1 1 @ @ @ @ @
8 | @ @ @ @ @ @ @ @
```

With the map above, `get_input(map)` should behave as follows.

```
(a) uncover a cell
(b) Mark or unmark a bomb
Please enter your option: a
There is not cells to be uncovered. Please try again.
```

```
Please enter your option: b
Uncover the cell at row: 3
                 column: 2
```

And a list `["b", 3, 2]` is returned.

Task 5 - `uncover()`

Implement function `uncover()` in the `Cell` class. The beginning of the assignment illustrates the behavior of this function. If the cell is a number, it uncovers itself. If the cell is a bomb, it uncovers all unmarked bombs. If the cell is a `.`, it uncovers all neighboring cells until reaching a number. This new version of `uncover()` creates a ripple effect when a `.` is uncovered, which in turns improves the gaming experience a lot.

Hint: Each cell stores all its neighboring cells in its `neighbors` attribute. A cell can uncover its neighboring cells by calling their `uncover()` method.

Task 6 - `wins(map)`

Implement a function `wins(map)` that takes a map and returns `True` if all non-bomb cells are uncovered and all 10 bomb cells are not uncovered. Otherwise, it returns `False`.

Task 7 - `loses(map)`

Implement a function `loses(map)` that takes a map and returns `True` if any bombs are uncovered. Otherwise, it returns `False`.

Task 8 - `save_game(map, filename)`

Sometimes the player wants to save the game, store it on a usb stick and finish it on another computer later. Implement a function `save_game(map, filename)` that takes a map and a filename string and writes to the file.

The file has 64 lines, representing the status of all 64 cells. The first line stores the cell at (1,1), the second line stores the cell at (1,2), and so on. In each line, a cell's `sign`, `uncovered` and `marked` are separated by a space.

Example 1

```
# # 1 . . 1 2 #
# # 2 . . 1 # #
2 @ 3 1 . 1 1 1
1 2 @ 2 1 . . .
```

```
. 1 2 # 1 . . .
. . 1 # 2 2 3 2
1 1 1 # # # # #
# # # # # # # #
```

The map above should result in the following file. Here we represent the input map as a string for readability but it is actually a list of lists of `Cells`.

```
1 False False
1 False False
1 True False
. True False
. True False
1 True False
2 True False
* False False
2 False False
* False False
2 True False
. True False
. True False
1 True False
* False False
2 False False
2 True False
* False True
3 True False
1 True False
. True False
1 True False
1 True False
1 True False
1 True False
2 True False
* False True
2 True False
1 True False
. True False
. True False
. True False
. True False
1 True False
2 True False
* False False
1 True False
. True False
. True False
```

```
. True False
. True False
. True False
1 True False
1 False False
2 True False
2 True False
3 True False
2 True False
1 True False
1 True False
1 True False
. False False
1 False False
* False False
* False False
* False False
1 False False
* False False
1 False False
. False False
1 False False
2 False False
3 False False
2 False False
```

Task 9 - `load_game(filename)`

Implement a function `load_game(filename)` that reads a file named `filename` and returns a `map` containing the information in the file. The given file that contains 64 lines in the correct format. The returned map contains `Cell` objects whose `sign`, `uncovered` and `marked` are set correctly. See the example in Task 9.

Task 10 - `main()`

Now write the `main` function for the game. The function keeps printing the current map, asking a valid move from the player, and applying the move until the player wins or loses. If the player wins, print to the screen:

```
Congratulations! You win!
Thank you for playing Minesweeper!
```

If the player loses, print to the screen:

```
Oh no! You lose!
Thank you for playing Minesweeper!
```

Then the function ends.

Task 11 - bonus task `load_game_smart(filename)`

Implement a smart `load_game_smart(filename)` that returns a complete map so that the player can resume the game. In addition to the description of task 9, `load_game_smart(filename)` further derives each cell's neighbors.