CSSE1001

Semester 2, 2013
Assignment 2
10 marks

Due Thursday 10 October, 2013, 9:30am

A GUI for Solving Mazes

1 Introduction

In assignment 1 you wrote a simple text-based maze solver. For this assignment you will write a GUI for solving mazes. A user of your program will be able to load a maze from a maze file or create a new maze on-the-fly. The user will then use the arrow keys to explore the maze. At the start only the tiles in the players immediate surroundings will be shown. As the player explores, more of the maze will become visible. At any stage the user can press the 'Quit' button to exit the program, press the 'Reset' button to start again (clearing the tiles that can not be seen from the start); or press the 'New' button to create a new maze.

2 Assignment Tasks

For each class and method that you write you need to provide a suitable comment giving a description and where necessary the type and any preconditions. You should use the triple-quote commenting style.

2.1 Download file

The file `assign2.py` is for your assignment. Add your name and student number in the space provided. When you have completed your assignment you will submit the file `assign2.py` containing your solution to the assignment.

The file already contains some code (in two parts). Do not modify any of this code! The first part should be left at the beginning of the file and consists of an import statement. The second part should appear at the end of your code. **NOTE:** You will loose marks if you don't follow these instructions.

2.2 Read the support file carefully

We have supplied you with some code in `MazeGenerator.py` to be used in your assignment. You need to understand how to make use of this code.

2.3 Write the code

Finally, write your solution to the assignment making sure you have included suitable comments. Your solution should include at least the following classes.

2.4 MazeApp Class

This class defines top-level of the application (i.e. the GUI).

2.5 Maze Class

This class is used to hold the maze data for the maze being solved and the position of the player in the maze. This class must define at least the following methods.

- `__init__(self, string)` that takes a string representation of a maze (as in assignment 1) and stores this in the object. You may use any of the code from assignment 1 to do this. This method must check that the string represents a valid maze and raises an `InvalidMaze` exception (which you need to define yourself) if any of the following occur:
  - there are no rows,
  - not all the rows are the same length,

  - the maze contains 'non-maze' characters,
  - there is more than one square representing the finish tile, and
  - the outside of the maze is not made up entirely of wall tiles.

  You do not need to check that the maze is solvable.

- `__str__(self)` returns a string representation of the stored maze suitable to be written to a file and later read back in as a valid maze.
- `move(self, direction)`moves the player according the the direction given by the key symbos for the arrow keys:
  (`"Left", "Right", "Up"", "Down"`).
- `reset(self)` moves the player back to position `(1,1)`.
- `get_size(self)` returns the pair (width, height).
- `get_tile(self, r, c)` gets the tile at position `(r,c)` (this is one of `' ', '#', 'X'`)
- `get_pos(self)` returns the position of the player.
- `is_solved(self)` returns `True` if and only if the player is on the finish tile.

Example:

```
>>> from assign2 import *
>>> fp = open('maze1.txt', 'U')
>>> string = fp.read()
>>> fp.close()
>>> m = Maze(string)
>>> str(m)
'#####\n#   #\n### #\n#X  #\n#####\n'
>>> m.get_size()
(5, 5)
>>> m.get_pos()
(1, 1)
```

```
>>> m.is_solved()
False
>>> m.get_tile(1,1)
' '
>>> m.get_tile(3,1)
'X'
>>> m.move('Right')
>>> m.get_pos()
(1, 2)
>>> m.move('Right')
>>> m.move('Down')
>>> m.move('Down')
>>> m.move('Left')
>>> m.move('Left')
>>> m.get_pos()
(3, 1)
>>> m.is_solved()
True
>>>
```

2.6 Examples

The course web page contains the following examples.

- A screenshots showing the GUI with the maze partially explored.
- A screencast showing the application in action.

2.7 Look and Feel

GUIs look different on different operating systems but you should at least satisfy the following criteria.

- The application should have a minimum size (`(300,300)`) and if the maze does not fill the space provided it should be centred. If it takes up more space then the application should expand to make all of the maze canvas visible.

- The canvas should be sunken.
- The new button should appear next to a spinner box on the right and be grouped together in a sunken frame.
- The File menu should contain 'Open Maze File', 'Save Maze File', and 'Exit' items.
- If an attempt is made to load an invalid maze string then an error box should pop up.
- When the player gets to the finish square a message box should pop up.

2.8 Hints

For saving and loading a maze use `tkFileDialog`.

To exit the application you can use something like `master.destroy()`.

Use `tkMessageBox` for both the error and completed messages.

To avoid creating lots of rectangles you might consider keeping track of where you have already drawn rectangles and use that information to prevent unnecessary redraws.

To move the player around you have two options. One is to delete the player circle and then redraw at the new location. The other option is to use `move`. In both cases you need to be careful that you don't hide the player circle under new rectangles. In the first approach you need to draw the player circle after you have drawn all the required rectangles. In the second approach you need to "lift" the circle to the top. This can be done by using the `tag_raise` method of the canvas after you have drawn the required rectangles.

Another approach is to draw all the squares in black and then use `itemconfigure` to change the fill colour. That way squares don't have to be deleted - just reset to black. There is also no problem with the player circle getting hidden if it is draw last. This may be the simplest approach.

3 Assessment and Marking Criteria

In addition to providing a working solution to the assignment problem, the assessment will involve discussing your code submission with a tutor. This discussion will take place in the practical session you have signed up to in week 12. You **must** attend that session in order to obtain marks for the assignment.

In preparation for your discussion with a tutor you may wish to consider:

- any parts of the assignment that you found particularly difficult, and how you overcame them to arrive at a solution;
- whether you considered any alternative ways of implementing a given function;
- where you have known errors in your code, their cause and possible solutions (if known).

It is also important that you can explain to the tutor how each of the functions that you have written operates (for example, if you have used a for loop or a while loop in a function, why this was the right choice).

Marks will be awarded based on a combination of the correctness of your code and on your understanding of the code that you have written. A technically correct solution will not elicit a pass mark unless you can demonstrate that you understand its operation.

We will mark your assignment according to the following criteria.

Criteria	**Mark**
Your code is mostly complete, correct, clear, succinct and well commented. You are able to explain your code.	8 - 10
Your code has some problems OR you have some problems explaining your code.	4 - 7
Your code is clearly incomplete, incorrect, too complex or hard to understand OR you have major problems explaining your code.	1 - 3
Your work has little or no academic merit.	0

A partial solution will be marked. If your partial solution causes problems in the Python interpreter please comment out that code and we will mark that.

Please read the section in the course profile about plagiarism.

4 Assignment Submission

You must submit your completed assignment electronically through Blackboard.

Please read
`http://www.library.uq.edu.au/ask-it/blackboard-assessment`
for information on submitting through Blackboard.

You should electronically submit your copy of the file `assign2.py` (use this name - all lower case).

You may submit your assignment multiple times before the deadline - only the last submission will be marked. After each submission please use Blackboard to check that the file you submitted was the one you intended to submit. Make sure the file is called `assign2.py` and not, for example, `assign2.py.py`

Late submission of the assignment will not be accepted. In the event of exceptional personal or medical circumstances that prevent you from handing in the assignment on-time, you should contact the lecturer in charge and be prepared to supply appropriate documentary evidence. You should be prepared to submit whatever work you have completed at the deadline, if required. Requests for extensions should be made as soon as possible, and preferably before the assignment due date.