CSCI 4041, Fall 2018, Programming Assignment 8

Due Tuesday, 10/30/18, 10:30 AM (submission link on Canvas)

This is not a collaborative assignment; you must design, implement and test the solution(s) on your own. You may not consult or discuss the solution with anyone other than the course instructor or TAs. In addition, you may not include solutions or portions of solutions obtained from any source other than those provided in class. Obtaining or sharing solutions to any programming assignment for this class is considered academic misconduct. If you are not sure what this means, consult the class syllabus or discuss it with the course instructor.

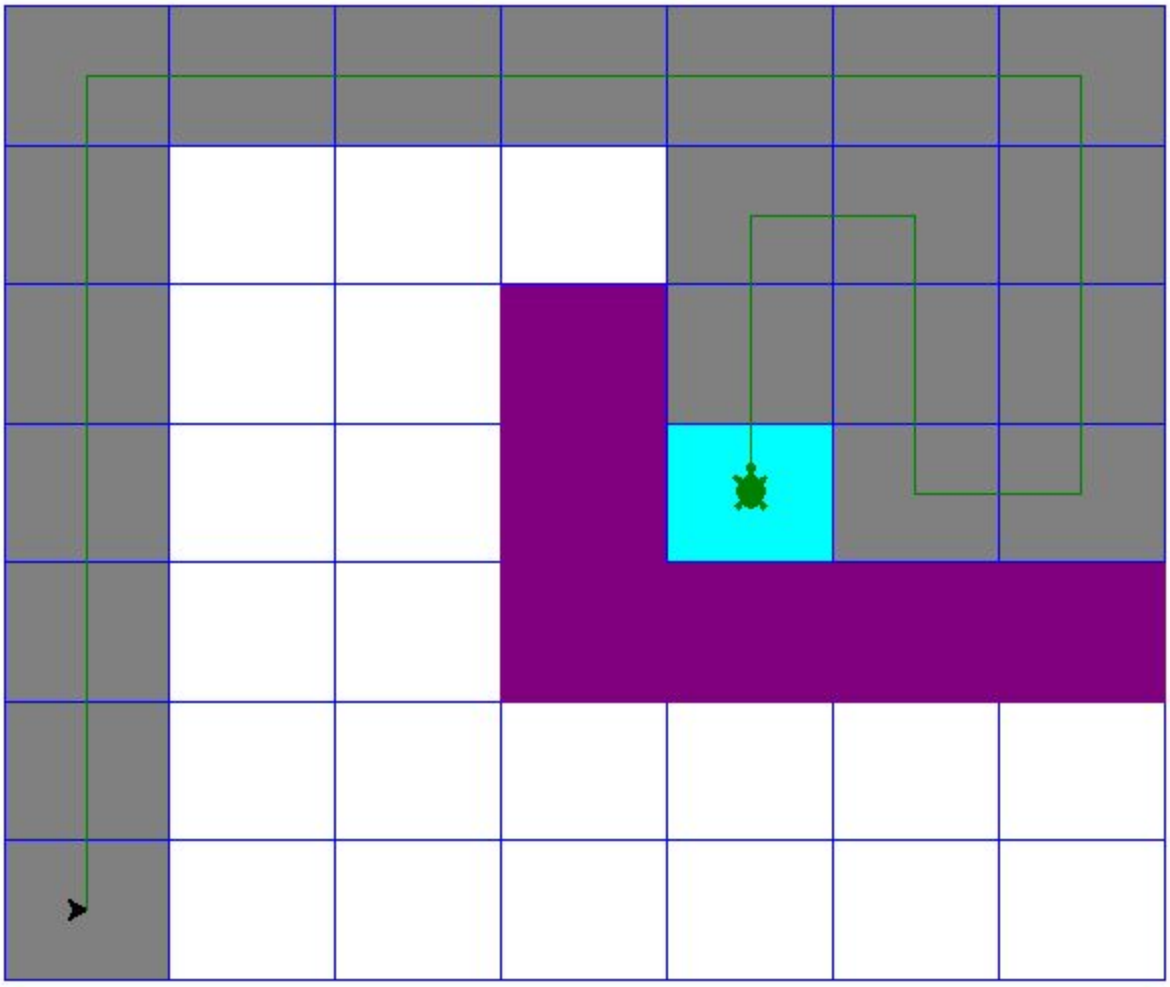
As the game you developed the turtle AI for last week moves into the beta testing stages, your AI has come under scrutiny. "The AI clearly knows too much about pathfinding, and it's making players feel inadequate", cites your boss, noting the turtle's uncanny ability to find the shortest path to its destination. You have been ordered to redesign the turtle's AI to find the goal using DFS rather than BFS, since this will still find a path to the goal, but not necessarily the shortest.

Aside from a snazzy new cyan color for water to make it more distinguishable from the purple death swamp, and a few new maps, the engine for the game (i.e. the template) has remained roughly the same: you still have a grid of nodes connected to their neighbors through the node's adjacency list instance variable .adj. See the Programming Assignment 7 write-up for details. Just like with BFS, you can and should use the built-in set_color accessor method for nodes to change the squares from white to gray to black as DFS searches them: this will make it far easier to visualize the algorithm.

Unlike the textbook version of Depth First Search (but like PA7), we want to stop searching once we find the goal node. This is made possible using the val instance variable in Square. val is an integer that takes on one of three values: swamp squares have a val of 0, normal squares have a val of 1, and the goal square has a val of 2. So, once you find the Square with a val of 2, the Depth First Search should halt, and you should determine the path of Square objects from the start to the goal and return it. Furthermore, the textbook DFS algorithm will search all nodes, including those inaccessible from the start node: you don't have to do this, since you can assume that there will be a path from the start to the goal (and there's not much of a point in finding the depth of inaccessible squares here anyway). Finally, note that we once again need to return not just the goal node but a list of nodes from start to goal, inclusive.

You should explore neighboring nodes in the order that they appear within the adjacency list of a given node: this will generally be up, then right, then down, then left. For example, you'll notice that in the search below (Test Case #3 in the template), the algorithm begins by searching nodes in an upwards fashion until it reaches the top of the grid, then goes to the right, and only then begins to consider squares located downwards or to the left, resulting in a very inefficient path to the goal. You'll also notice that there are no black nodes in this case: that is due to the fact that the algorithm reached the goal without hitting a dead end and backtracking

in this scenario, so all nodes explored form one long path. This will not necessarily be true in general.



Download the PA8.py template from the course website. The template contains the Square class, a set of helper functions primarily for setting up the grid and drawing it, and a series of test cases. You must implement the `find_path` function, which takes as input the Square object representing the lower left square (0,0), and outputs a list of Square objects that specify the shortest path of Squares from the start to the goal, inclusive and in that order.

Requirements:

- You must download the template file PA8.py and edit the `find_path` function. You can create your own helper functions, but don't edit the code beyond the "DO NOT EDIT" line.
- The code provided takes the starting node and colors it gray. You must complete the functionality of `find_path` by using DFS to find a path from the start to the goal node that does not pass through the swamp, prioritizing movement up, then right, then down, then left (relative to the overall grid, not the turtle's current heading); and then return that

path in the form of a list of consecutive Square objects that the turtle can move through to go from the start to the goal.

- Your program must run without errors on the version of Python installed on the CSELabs machines, Python 3.5.2. (if you're testing this on CSELabs, you need to type python3 or idle3 instead of python or idle to start the correct version from the terminal)
- You are not permitted to use any built-in Python sorting routines like the `sorted()` function or the `.sort()` list method. You are also not allowed to use any Python function that asks for user input, such as `input()`, as this will break the grading script.
- You must implement the Depth First Serach algorithm found in Chapter 22 of the textbook. Any other algorithm will receive very little credit.
- However, note that while the textbook algorithm describes how to conduct a full Depth First Search, computing the depth and previous node for every node in the graph. Your version must stop when the goal node is reached, and must then return a list representing the final path from the start to the goal.
- This assignment will be graded automatically based on the number of test cases your program passes. There will be several secret test cases in addition to the ones included in the template to ensure you're not hard-coding in the solutions.
- This program will only run test cases until you fail one, avoiding the problem of having to scroll through test output to find the one broken test case.
- The grading breakdown for this assignment is as follows:
    - 30%: File runs without syntax errors
    - 70%: Passing test cases without breaking any requirements.
- The unedited template file already runs without syntax errors. This means that if your program causes syntax errors, you will get a better score by just submitting the original template unedited.
- Submit your edited PA8.py file to the Programming Assignment 8 link on Canvas before 10:30 AM on 10/30/18. No credit will be given for late submissions.