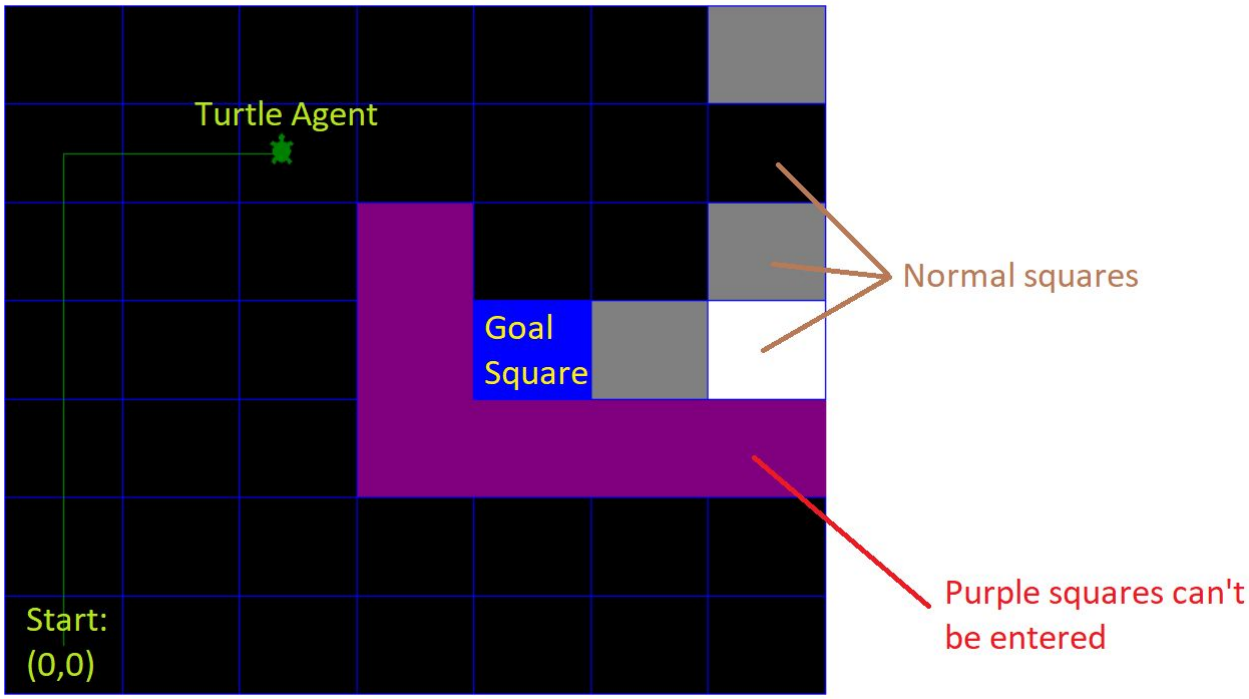
CSCI 4041, Fall 2018, Programming Assignment 7

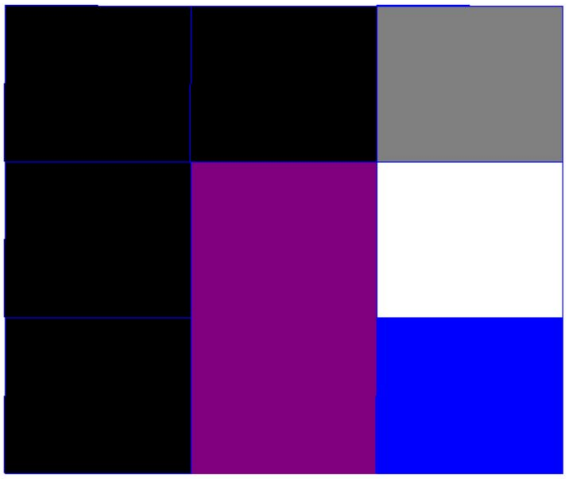
Due Tuesday, 10/23/18, 10:30 AM (submission link on Canvas)

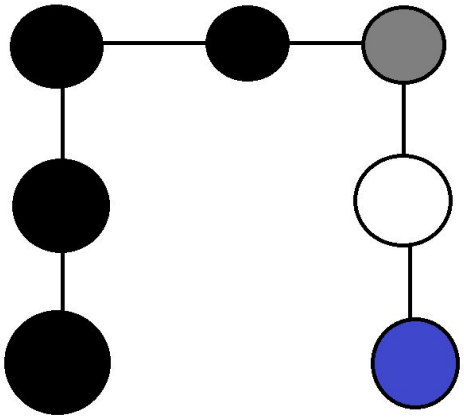
This is not a collaborative assignment; you must design, implement and test the solution(s) on your own. You may not consult or discuss the solution with anyone other than the course instructor or TAs. In addition, you may not include solutions or portions of solutions obtained from any source other 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.

You are programming the pathing for a turtle AI in a video game: the turtle is trying to avoid danger and make it to the water as quickly as possible. Specifically, the turtle is placed within a grid of squares, starting at position (0,0) in the lower left. Each square is either a deadly purple swamp square, which should not be entered, a normal square, which can be white, gray, or black, or the blue goal square. The turtle must move on the grid, and can only move one square up, down, left, or right at each step: the objective is to do this in as few steps as possible, so we are finding the shortest path from the start to the goal.



The grid data is given to the turtle in undirected graph form; that is, the turtle does not know where the goal is, and for a given square can only fetch the adjacent squares that aren't swamps (not even data can information can pass through the swamp). Below is shown an example grid and its undirected graph form.





Notice that the swamp squares do not appear in the graph form, because they do not appear in the adjacency lists of the other nodes. This makes it possible to find the shortest path from the start node to the goal node using Breadth First Search: the minimum depth path to the goal node is guaranteed to be the shortest path that doesn't go through the swamp. Each node in the graph is implemented in Python as a Square object, which includes instance variables such as the coordinates of the square within the grid, but also BFS node data such as the adjacency list for the node, the node's depth within the search, and a pointer to the previous node so that the path can be reconstructed at the end of the search.

Since we are allowing the squares to be colored white, gray, or black when they aren't the swamp or the goal, we can just use the BFS color variable as the actual color for the node: this means that squares should change from white to gray to black as they are searched by the algorithm. Note that color is a private instance variable, so unlike every other instance variable, you must use the provided getter and setter methods to access it (this also ensures that the color of the square is changed visually when it is changed within BFS). The goal node is a bit of a special case: technically its color instance variable is set to white to be consistent with BFS, but when it is actually drawn it is shown as blue for visual clarity.

Unlike the textbook version of Breadth First Search, 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 Breadth First Search should halt, and you should determine the path of Square objects from the start to the goal and return it.

Download the PA7.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 increasingly complex 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 PA7.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 BFS to find the shortest path from the start to the goal node that does not pass through the swamp, 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 Breadth 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 Breadth 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 PA7.py file to the Programming Assignment 7 link on Canvas before 10:30 AM on 10/23/18. No credit will be given for late submissions. Do not submit the test files.