

Project 5: Turtle Maze Solving

DCS 229

Winter 2026

In this project you will implement the Stack data structure and use it to implement the Depth First Search algorithm.

Learning Outcomes:

- Implement the Stack data structure.
 - Use your Linked List and Stack implementations to solve programming problems.
 - Write code that interfaces with existing Python modules
 - Use unit tests to verify the correctness of an implementation.
 - Develop professional programming practices.
-

1. Complete the [Stack.py](#) starter code from Lyceum.

- You need to use your LinkedList implementation from Project 4.
- You should not change the definitions of any functions.
- You should include unit tests in the main() function to verify that your stack works correctly.

2. Get to know the TurtleMaze code

- Read [Exploring a Maze](#) (section 5.11 from *Problem Solving with Algorithms and Data Structures using Python*) and work through the example code.
- Look through the [TurtleMaze.py](#) file from Lyceum. This code is a modified version of the turtle maze code from section 5.11.
- Experiment with the TurtleMaze class by adding code to the main() function. Some ideas:
 - Make a TurtleMaze object with a different maze file (see the .txt files on Lyceum)
 - Print the contents of the cell in the top right corner of the maze
 - Update the turtle's position
 - Move the turtle horizontally and change the color of the dot trail it leaves
- Come to office hours early to ask questions!

3. Get familiar with the depth first search algorithm

- Read through [this webpage on depth first search](#) and try the interactive demo [note: you will be implementing the non-recursive version by directly using a Stack data structure]
- Try these interactive examples:
 - <https://www.cs.usfca.edu/~galles/visualization/DFS.html>
 - <https://visualgo.net/en/dfsbfbs>

- c. Test the depth first search algorithm on paper with the turtle mazes. (see page 3)
- d. Come to office hours early to ask questions!

4. Implement a non-recursive depth first search algorithm to solve the turtle mazes.

- a. Create a new python file named [TurtleDFS.py](#) and import your [Stack.py](#) code and the [TurtleMaze.py](#) code.
- b. Write a function called `get_neighbors()` that takes in a maze object and a cell object and returns a list of non-obstacle cell objects that are north, south, east, and west of the current cell.
- c. Test your `get_neighbors` function in your `main()` function
- d. Write a function called `dfs()` that takes in a maze object. This function should run the depth first search algorithm on the maze, using a Stack object. Your code will use the start and goal from the TurtleMaze object. Use the `updatePosition` function to move your turtle through the maze. Your function should return True if the turtle reaches it's goal, and False if it does not.
- e. Implement backtracking so your turtle doesn't teleport in the maze. That is, your turtle is only allowed to move to a neighboring cell.
- f. Test your search function on a variety of mazes!

To submit your project:

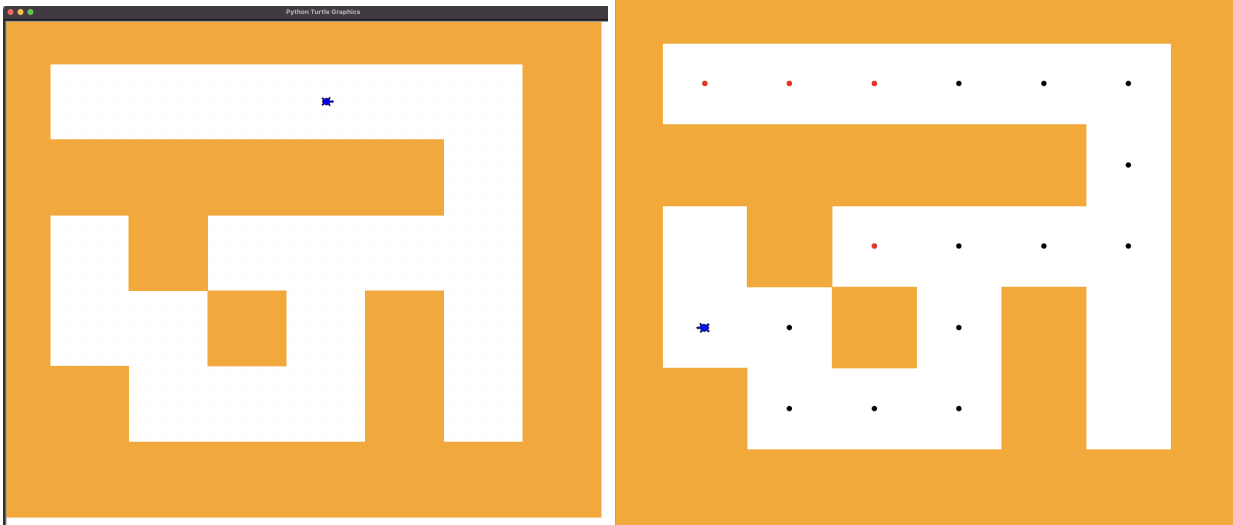
Upload 3 python files named: [LinkedList.py](#) [Stack.py](#) and [TurtleDFS.py](#) to Lyceum. Bring a printed copy of these files to class Please print your code from the pycharm IDE.

Grading:

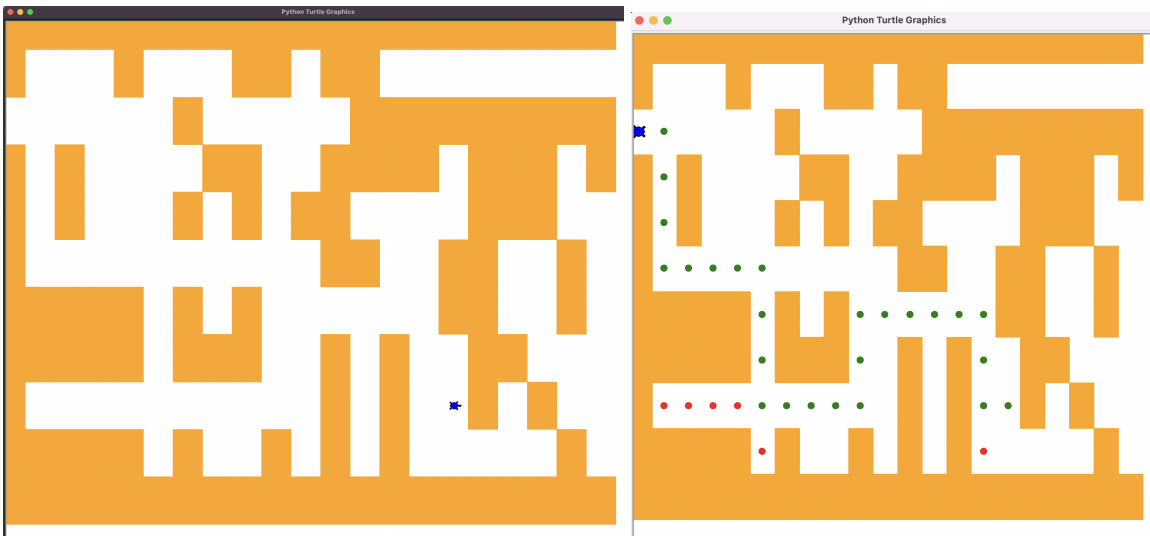
I will assign you a score of **Satisfactorily Complete** (SC), **Needs Work** (NW), or **Unacceptable** (U) for each factors:

1. Code Correctness & Testing – Does your code work according to the project description?
2. Code Readability & Documentation – is your code written in a way that is readable for a human? This includes logic variable names and clear formatting. Is your code well documented? Do you have comments, typehints and docstrings?
3. In-class code annotation – Did you demonstrate that you understand your code during the in-class annotation? Can you explain and answer questions about your code?

maze_1.txt



maze_2.txt



Maze_3.txt

