CSCI 232: Data Structures: Project 2

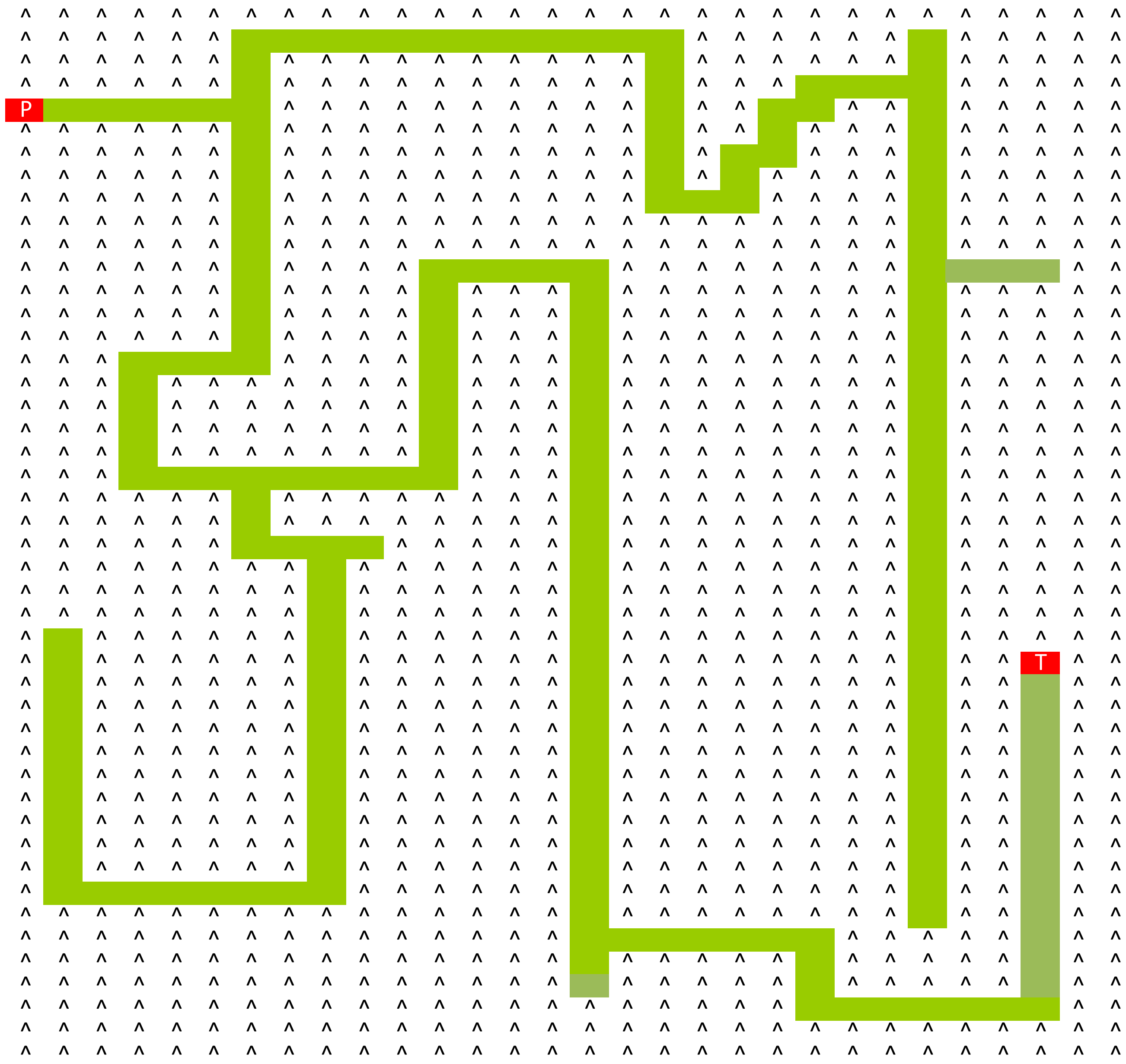
Due Wednesday, October 26 @ 11pm

Problem Description

Consider a problem whereby you are given a representation of intersecting trails up a mountain, and you must help a hiker find her way from the parking lot (P) to the top of the mountain (T). Program a solution to this general problem that adheres to the following requirements:

- Your program should read in a simple text file containing rows of characters, where the '^' character represents a tree that block the trail, a blank space is open trail, 'P' for the parking lot, and 'T' for the top of the mountain. This file can be of any size and any representation. Each row of the text file will contain a string of characters; the only spaces in the file will be for open trail. For example, the first 2 rows of the text file for the following representation would look like:

```
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^            ^^^^^^ ^^^^^
```



Program specifications continued…

- After reading in the text file, your program should create a visually appealing and creative graphical representation of the problem using turtle graphics.
- Program a solution to the puzzle whereby the "hiker" starts at the parking lot ('P'), and makes his or her way to the top of the mountain ('T'). You must **use a stack** to solve this puzzle, **NOT** recursion.
- As the hiker makes her way up the mountain, show her movements in the turtle graphic. Make sure the movements are paced well so that the user can easily see what's happening. For example, wait a few seconds for the hiker to begin, and if the hiker tries to move into a tree barrier, perhaps that tree location should turn a different color to indicate that it's been tried or visited.
- Your hiker should not be revisiting any locations that have been previously visited.
- Your program solution should work on any trail/mountain representation as long as there is a solution (i.e., a trail that gets the hiker from the parking lot to the top of the mountain).
- Call your program *lastname*_hiker (where lastname is your lastname), and allow us to run it using the following command, where "mountain.txt" is any text file representing the mountain and trails.

  >> *lastname*_hiker("mountain.txt")

Grading. Your program will be graded according to the following criteria:

- Is it programmed according to given specifications?
- Does it work with any input text file as long as the format is correct?
- Is it saved with the proper name, and does it run with the proper command?
- Is the turtle graphics representation appealing and creative? Does it seem like a lot of effort went into this part of the assignment?
- Does the graphic show the movement of the hiker as she walks the trails? Are the movements paced well so the user can easily see what's happening. For example, if the hiker tries to move into a tree barrier, what happens? How can the user see this isn't a way forward? How can the user see what locations have been visited already?
- How can we tell if/that the program is working correctly and if the hiker found her way to the top?
- Does the program work correctly? Given a starting position 'P' and and ending position 'T', does the hiker find her way every time, assuming there is a solution? What happens if there is not a solution?
- Does the program use a Stack and NOT recursion to solve the puzzle?
- Is the code well commented and readable?
- Is the code well organized and modular?
- Is the code efficiently written?

Helpful Hints:

- You may wish to represent the mountain as a list of lists. Each row of characters you read in from the input text fill can be a list of characters; the entire representation/problem space can be a list of rows. See *sample_mountain.py* as an example.
- Your stack should hold states or locations of the grid. If a state or location has not been visited previously and if it is not the ending state, then all unvisited adjacent states will be pushed onto the stack for subsequent examination.
- Start early, and do not hesitate to seek early feedback from me and/or Alex.
- Your final submission should represent ***your*** understanding and coding of the problem, not anyone else's.