ENGR 101 - Introduction to Programming

Mini Project 2

In this mini project, you are going to develop a basic maze solving and treasure hunting game. The important goal of this project is to practice functions, for loops, while loops, operators, using break and continue in the loops. The details about the game that you are going to develop are provided below. (Note: example movements of turtle in the game will be posted as GIF images and screenshots of program)

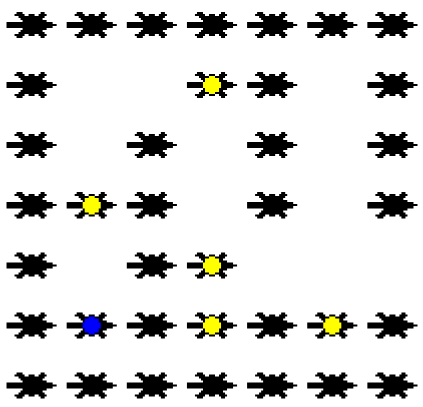
The Logic and The Parts of Maze Solver Game:

In this project you will be given a maze of a particular structure and a turtle within the maze. The objective of the game is to assist the turtle so that it can escape from the maze and/or pick up treasures located in the maze.

The maze contains two distinct items:

- Treasures: shown as yellow turtles, these items can be picked by a turtle. You will be given functions to detect and pick up a treasure.
- Walls: shown as black turtles, your turtle should not pass through a wall. You will be given a function to understand if a turtle is facing a wall or not.

An example maze is shown below:



Files:

The game is designed by using **Swampy** module of python 3. You will be given the following files:

- wm.py: contains the module WorldManager that can create a maze containing walls and treasures within the TurtleWorld. The WorldManager reads a maze from a txt file. You do not have to worry about the details regarding the WorldManager. You should not modify this file.

- maze1.txt, maze2.txt, maze3.txt, maze4.txt: contain mazes for questions 1, 2, 3, and 4 respectively. You may edit any of the files to create your own maze.
- main.py: the file that you will run for the game. You will implement the q1, q2, q3, q4 functions within this file.

API For WorldManager file

The functions you will **need** to use from the WorldManager module is given below:

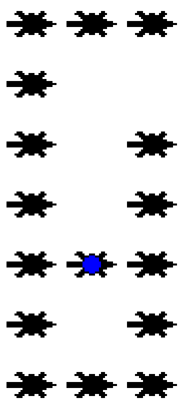
1. **is_facing_wall(turtle):** Given a turtle, the function returns True if the turtle is facing a wall. Returns False otherwise.
2. **is_over_treasure(turtle):** Given a turtle, the function returns True if the turtle is over a treasure. Returns False otherwise.
3. **pick_treasure(turtle):** Given a turtle, the function picks the treasure located under the turtle. Treasure is removed from the maze. If such a treasure does not exist, this function prints an error message. Make sure that the turtle is over a treasure before calling this function.

Settings

You have to guide the turtle in the following settings. General rules for game:

- Important: our smallest unit of operations is 30 pixels. (E.g, you should move by calling fd(turtle, 30))
- You should not go over walls (e.g., black turtles). Doing so for a run means, you failed in that run.
- You should exit from right most column
- Columns are always 1 unit wide. See below by what we mean by column.
- Turtle always starts in the left-most column pointing towards east.

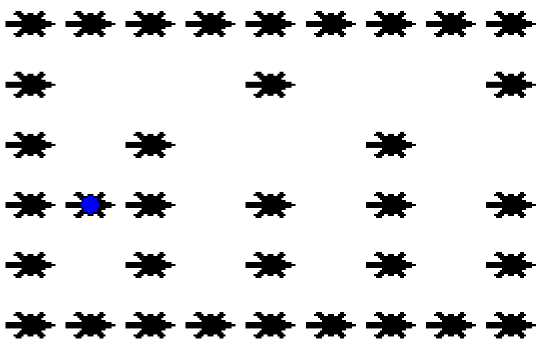
Setting 1



In our context a column is a one unit wide vertical road surrounded by wall segments from east, west, south, and north. Each column in this project will have a single exit point on the east side of the walls.

In this first setting, the maze will always have a single column, with a single exit in the east side. The turtle can start in any part of the column facing east. The exit can be located anywhere on the right wall. The objective of the turtle is to leave the maze from the exit.
Note that the turtle does not know where the exit is, thus needs to go forward along the column looking for the exit. If you start searching for the exit by going first up and the exit happens to be down, you will have to go back to find the exit.
Hint: use a while loop and check continously if the right hand side contains a wall segment (e.g., is_facing_wall). Be careful not to pass over the north-most or south-most walls.
You need to implement function q1 for this setting.

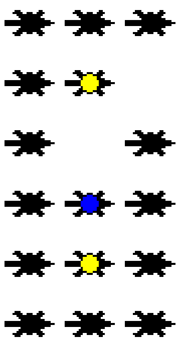
Setting 2



In this setting, we have multiple columns. The number of columns will be an input to the program. The objective is the same, the turtle should leave the maze through the east-most exit.

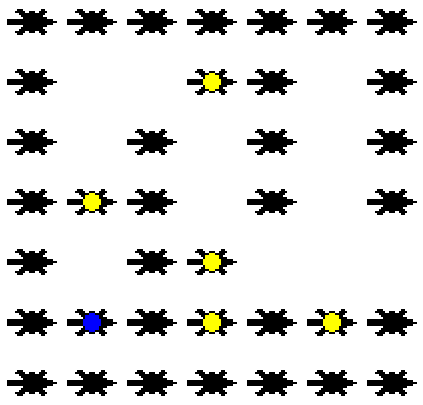
thus will get it as a parameter to the function q2 you will be implementing.
You need to implement function q2 for this setting. Number of columns will be an input parameter to the function.
Note: Call q1 within q2 iteratively to easily achieve the objective.

Setting 3



This setting is similar to the setting 1 except the column now may contain treasures. We assume the turtle has a carrying capacity. The maximum number of treasures that can be carried by the turtle will be an input to the program. The objective is to pick all treasures within the column as long as it is possible. If the turtle has sufficiently large amount of capacity to carry all treasures, the turtle should leave the maze as described in setting 1. Otherwise, the turtle should stop as soon as the carrying capacity is full without leaving the maze.
When turtle is done with this setting, it should print the number of treasures it has collected on the console.
You need to implement function q3 for this setting. The maximum no of treasures will be an input parameter to the function.

Setting 4



Setting 4 is very similar to setting 3 except we have many columns. The maximum number of treasures that can be carried by the turtle and the number of columns will be an input to the program.
Again when turtle is done with this setting, it should print the number of treasures it has collected on the console.
You need to implement function q4 for this setting. The maximum no of treasures and the number of columns will be input parameters to the function.

You will be given a demo of the program that shows an ideal execution. You do not need to follow the exact strategy used in the demo. As long as, you fulfill all objectives you will get full credits.

Warnings:

- **Do not** talk to your classmates on project topics when you are implementing your projects (This is serious). **Do not** show or email your code to others (This is even more serious). If you need help, talk to your TAs or the instructor, not to your classmates. If somebody asks you for help, explain them the lecture slides, but do not explain any project related topic or solution. Any similarity in your source codes will have **serious** consequences for both parties.
- Carefully read the project document, and pay special attention to sentences that involve "**should**", "**should not**", "**do not**", and other underlined/bold font statements.
- If you use code from a resource (web site, book, etc.), make sure that you reference those resource at the top of your source code file in the form of comments. You should give details of which part of your code is from what resource. Failing to do so **may result in** plagiarism investigation.
- Even if you work as a group of two students, each member of the team should know every line of the code well. Hence, it is **important** to understand all the details in your submitted code. You may be interviewed about any part of your code.

How and when do I submit my project? :

- Projects may be done individually or as a small group of two students (doing it individually is recommended for best learning experience). If you are doing it as a group, only **one** of the members should submit the project. File name will tell us group members (Please see the next item for details).

- Submit your own code in the **main.py** Python file; Rename your code file with your and your partner's first and last names (see below for naming). Do not submit the other project files. We will use our own maze files to grade your project.
    - If your team members are Deniz Barış and Ahmet Kemal Çalışkan, then name your code file as deniz_baris_ahmet_kemal_caliskan.py (Do **not** use any Turkish characters in file name).
    - If you are doing the project alone, then name it with your name and last name similar to the above naming scheme.
    - Those who **do not** follow the above naming conventions **will get 5 pts off** of their grade.
- Submit it online on LMS by **empty**

Late Submission Policy:

- -10%: Submissions between 17:01 – 18:00 on the due date
- -20%: Submissions between 18:01 – midnight (00:00) on the due date
- -30%: Submissions which are 24 hour late.
- -50%: Submissions which are 48 hours late.
- Submission more than 48 hours late will not be accepted.

Grading Criteria? :

Code Organization			**Functionality**			
Meaningful variable names (3 pts)	**Proper use of functions, compact code with no unnecessary repetitions (4 pts)**	**Sufficient commenting (4 pts)**	**Problem 1(40 pts)**	**Problem2(20 pts)**	**Problem 3 (30 pts)**	**Problem 4 (20 pts)**

Have further questions?:
Please contact your TAs if you have further questions. If you need help with anything, please use the office hours of your TAs and the instructor to get help. **Do not walk in randomly (especially on the last day) into your TAs' or the instructor's offices. Make an appointment first. This is important. Your TAs have other responsibilities. Please respect their personal schedules!**

Good Luck!