6.177 Problem Set

Langton's Ant Zero-Player Game

due 11:59pm, January 15, 2016

Overview

For this problem set, you will complete a brief warmup exercise and then implement your own version of Langton's Ant. This problem set is due at 11:59pm on January 15th. However, if your submission does not pass our tests, we will allow a second submission due 11:59pm on January 20th. Remember that a passing grade on the problem set is required to pass the class.

Langton's Ant is a famous ”zero-player” game. It runs on a two-dimensional grid of squares and exhibits some very interesting emergent behavior based on a simple set of rules.

The ant initially starts on a grid of white squares. On every step, the ant progresses according to two rules:

1. If the ant is on a white square, turn 90° right, flip the color of the square to black, move forward one unit
2. If the ant is on a black square, turn 90° left, flip the color of the square to white, move forward one unit

This progression continues indefinitely.

For this problem set, you will use the `pygame` package. Many of you will choose to make games for your final project, and `pygame` is a great package to use (and the one that the staff will officially support). You will need to go to `http://www.pygame.org/download.shtml` to download `pygame`. If you are using a Mac, we have some additional helpful hints for you on Stellar under the “Materials” section.

This problem set may be less straightforward coding and more reading documentation than you were expecting - this is intentional, and is very similar to the format of many of the labs that you will see in 6.01. If you have any questions, feel free to ask a TA.

Part 0 - Your Information

Open up `runAnt.py` and fill in your name and your e-mail address in the header.
DO NOT FORGET TO FILL THIS IN.

Part 1 - Warmup

Read the three warmup exercises in `runAnt.py` and fill in their respective methods. To test them, uncomment the function `test_warmup()` at the bottom of the file and run the file.

Part 2 - The Board

Basics

We're going to start out by defining some global variables in order to simplify the rest of our coding.

Our game board will be split into "squares" rather than using normal pixel measures. Find the place where we define the `width` and `height` of our squares and pick a good value for each one. We recommend looking at your ant image (`ant.png`) to pick your values.

Your board starts at the top left pixel of your window. There are two procedures, `get_row_top_loc` and `get_col_left_loc`, that give the top left pixel of a given square. We're going to make a 10 pixel border around the entire window. Fill in these procedures, using the global `width` and `height` variables that you defined in the previous step. Remember: your rows and columns are 0-indexed, so the top left pixel of square (0, 0) is located at (10, 10).

Drawing your grid

Read the `update_text` and `new_game` methods and make sure that you understand what they're doing.

The `update_text` method will write the number of steps that the game has progressed on each step. The `new_game` method will initialize the game and set the initial size of the window.

Note that we're using the `size` variable to make decisions about how large the game window is. What is `size`? Ask a TA if you aren't sure.

Skip down to the `draw_grid` method, which takes a `screen` (the surface to draw on) and `size` as arguments. This will draw lines on the window to define the grid. Fill it out.

You may find the method `pygame.draw.line` useful. Note that `start_pos` and `end_pos` should be (x,y) tuples.

When you're done, you can test that the grid will be drawn properly by uncommenting the `test_part_two()` function at the bottom of the file.

Part 3 - The Pieces

Squares

Find the `Square` class and fill in the values for the three missing properties in the `__init__()` method. Also fill in code for the `get_rect_from_square` and `flip_color` methods. Don't forget - you need to refill your squares with the new color using `self.image.fill(self.color)` if you want to see the new colors!

Board

Find the `Board` class.

The `Board` class has one pygame "Sprite" that represents all of the squares. Sprites are a very powerful encapsulation that you will learn about during the `pygame` lecture, but for now, don't worry about them too much.

You will want to create a new white `Square` object for each square on the board and store them using an appropriate data structure. We use the variable name `boardSquares` for this data structure. After you add each square to `boardSquares`, you will also need to add it to `squares`. If you name each square `s`, you will need to call `self.squares.add(s)` after creating each square `s`.

Write the `get_square` method as well to retrieve a square (x, y) from `boardSquares`.

To test part 3, uncomment the `test_part_three()` function. The squares will be drawn slowly one at a time, and then the grid will be drawn over them. Don't worry if the squares draw out of order.

Part 4 - The Rules

The Ant

Find the `Ant` class.

Fill in definitions for the four parameters in the `__init__()` method that are currently `None`.

Fill in `get_current_square` so that it returns the Square that the Ant is currently on.

We use the `self.rotation` parameter to describe the (x, y) change when the ant takes its next step. For example, the ant starts out pointing up, which means that it will move 1 row up and 0 columns to the left or right on its next step. Therefore, our `self.rotation` parameter is intially (0, 1).

Fill in the `rotate_left` and `rotate_right` methods. You may find the `pygame.transform.rotate` method useful.

Movement

Find the `rotate_ant_get_square` method and fill it in.

This method should rotate the ant according to the color of the square that it's on. Don't forget to return the ant's current square at the end!

Find the `step_forward` method and fill it in.

This method should cause the ant to take a step forward in whatever direction it's currently pointing (your `rotation` parameter may come in handy). Don't forget to update the ant's relevant parameters at the end of this method!

It's OK if your game stops or throws an exception upon hitting an edge. But if you find that your ant tries to go off of the side of the game board, you should recheck your implementation of this method.

Part 5 - The Game

Look at the `main_loop` method. Use the methods that you wrote in the other parts of this problem set to fill in the missing lines (denoted with **).

Once you're done, you should be able to run the file, and your ant should move around the screen!

Part 6 - Challenges

If you have correctly completed the entire problem set up until this point and turn it in as-is, you will receive a grade of `4/5`.

In order to receive a `5/5`, you must implement (at a minimum) the following 3 changes to impove the functionality of your game.

1. Allow the user to specify the dimensions of the game board. (Bonus: Allow for rectangular board layouts, i.e. `5x12`)
2. Handle the case when the ant hits an edge. The ant should wrap around to the other side of the board and continue (think pac-man or asteroids).
3. Randomize the initial starting position and orientation of the ant.

If you find yourself stuck on these challenges, a TA can help point you in the right direction (don't wait until the last minute!) or you can check Stack Overflow for python how-tos.

Problem set created by Karen Sittig & Colleen Rock.
Additions and revisions by Keanu Delgado.
Revised January 11th, 2016.