Programming Project 1: Snake

CS-106 Fall 2020

Overview

GitHub Repository: https://github.com/Qu-CMU/ACES_CS106_F20_Snake_Lab

This lab is designed to test your capabilities. The tasks will increase in difficulty and provide have fewer hints. If you get stuck, don't be discouraged and ask for help. Future labs will be calibrated based on the classes experience with this one.

1. Make a GitHub Account
2. Clone the Repository
3. Background Information
4. Instantiating Objects
5. Getting the Snake Moving
6. Eating and Moving Food
7. Growing the snake
8. Collision checking
9. Optional Challenges

Make a GitHub Account

- Go to https://github.com/ and sign up for a free account.
- Download and install the free Github desktop app from https://desktop.github.com/

Clone the Repository

- Clone the project repository from https://github.com/Qu-CMU/ACES_CS106_F20_Snake_Lab

Background Information

Snake is a very simple game where the player controls a snake. The objective of the game is to collect food. However, this causes the snake to grow and if the player collides with the wall or with their own body, the game is over. https://www.google.com/search?q=play+snake

Snake is a grid-based game, meaning there are distinct "tiles" where an object can be.

(0, 0)	(1, 0)	(2, 0)	(3, 0)	(4, 0)	(5, 0)
(0, 1)	(1, 1)	(2, 1)	(3, 1)	(4, 1)	(5, 1)
(0, 2)	(1, 2)	(2, 2)	(3, 2)	(4, 2)	(5, 2)

Figure 1: Example of a grid with 3 rows and 6 columns.

As you might have noticed, the origin is in the top-left. This is because in computer graphics, the origin is in the top-left. The x-value increases as you move to the right of the screen and the y-value increases as you move down the screen.

In Snake, you can only move the snake up, down, left, and right. The direction that the snake is moving is called its heading. Heading is written as (x_velocity, y_ velocity) where:

$$new\_x\_position = current\_x\_position + x\_velocity$$

$$new\_y\_position = current\_y\_position + y\_velocity$$

At the top of the script there are several heading constants defined. Using Figure 1, pick a position and test if the constants UP, DOWN, LEFT, and RIGHT move your position in the correct direction.

There are several other constants that may be useful. The color constants represent colors using RGB values (R, G, B). https://www.rapidtables.com/convert/color/rgb-to-hex.html allows you to move sliders to see how the RGB values translates to colors.

The WINDOW_WIDTH and WINDOW_HEIGHT constants define the size of the game window in pixels. TILE_DIMENSION specifies the size of each tile in pixels. You can change these values, but make sure that TILE_DIMENSION can evenly divide WINDOW_WIDTH and WINDOW_HEIGHT otherwise you will have part of the tiles cutoff on the edge of the screen.

Instantiating Objects

- The first thing we need to do to get our program working is to create a **Snake** object and a **Tile** object. If we look at the __init__() function we can see what arguments a tile object and a snake object takes.
- If we do this correctly, when we run the program, a window should appear that looks something like Figure 2.

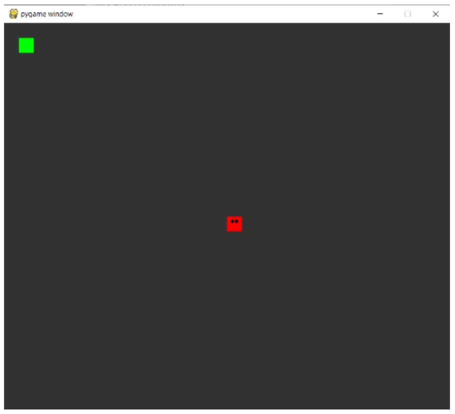

Figure 2: Display snake and food

Getting the Snake Moving

- Now we need to make the snake move. To do this, we need to complete the **move**(heading) function in the **Tile** class.
- Don't overthink this one. If you get stuck, read the background information (but you should have read it before you started).
- Once **Tile.move** () has been written, the snake should begin moving when you press a key.

Eating and Moving Food

- For the game to progress, we need the snake to be able to eat food. To do this, we need to check if the snake and the food share a position in **SnakeGame.snake_on_food**(). (Think about which part of the snake we should check.)
- To make the food move, we need to be able to generate a random (x, y) position to move the food. We need to complete **SnakeGame.randomPosition**() to make it return random positions instead of (1, 1).
- If both **snake_on_food**() and **randomPosition**() have been implemented correctly, the food should now move when the snake touches it. If you are having trouble, try implementing **randomPosition**() first. You can check if it works by seeing if the food is in a different place every time you start the game.

Growing the Snake

- Finally, we need to make the snake grow. To do this we need to create a new tile and add it to the end of the snake. Think about how you can figure out the position to add the tail and what direction to set it.
- This one might be a little tricky, if you get stuck try asking for help on Google Classroom.
- Now your game is almost complete! You should be able to play snake, except you can't lose!
- Weird things might happen if you intersect yourself.

Collision Checking

- Now we just need to make the game a challenge. We need to implement two functions, **SnakeGame.check_bounds**() and **SnakeGame.game_over**()
- **SnakeGame.check_bounds**() should see if the snake has travelled off the screen.
- **SnakeGame.game_over**() should check if the snake has touched itself.
- Congratulations, you did it! You now have a working game of snake.

Optional Challenges

- So, you thought the lab was too easy?
- Make the world round. If the snake goes off the screen on one side, it wraps around to the opposite side. You can do this by replacing the code in **check_bounds**().
- Can you think of a way to play the game that guarantees you cannot lose until you reach maximum length? (Assume you programmed an AI that makes moves perfectly.)