Introduction to Computer Science I
MCS 177 - Fall 2019
Project 10: Snake
Due 12/13

Overview

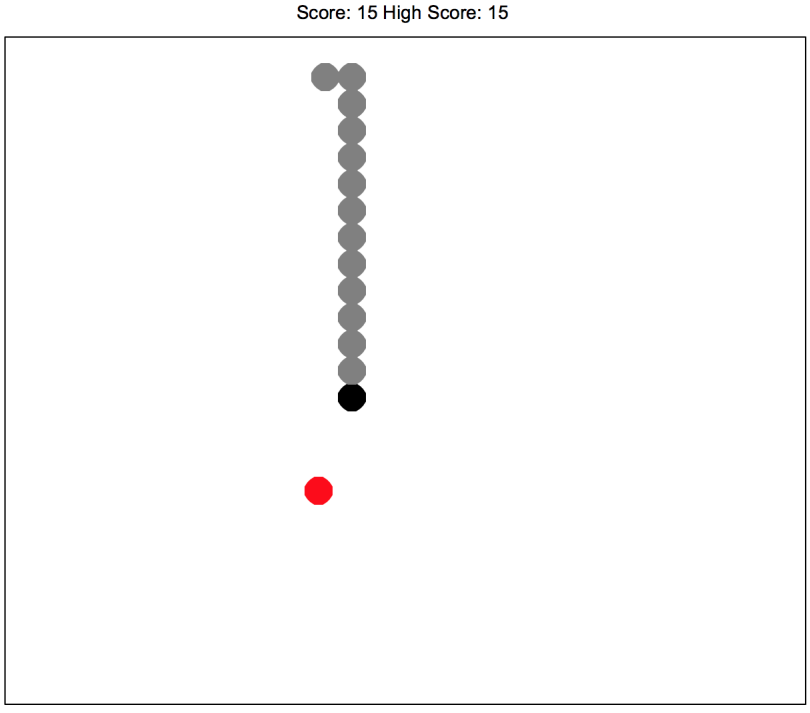
For this project, we will implement the classic arcade game *Snake* in Python using the cTurtle module. You can play Google's version of the game at the following link.

`https://www.google.com/search?q=play+snake`

Your version of snake won't be as nice looking as Google's, but it will have the same functionality! Here are some notes about how the game works.

- You can change the direction of the snake with the arrow keys.
- The snake keeps moving in the same direction, unless you change the direction.
- There is always an apple in the window, and the goal is to eat as many apples as possible before losing.
- If the head of the snake runs into the body of the snake, or the edge of the window, the player loses, and the game resets.
- Every time the snake eats an apple, the snake gets longer by one unit, the score goes up by 1, and the apple moves to a random position in the window.
- In addition to keeping track of the current score, the game keeps track of the highest score a player has achieved.

Below, we include an example of how our snake game might look.



In order to implement our game as a well-organized program, we will use classes and the principles of object-oriented design.

You may choose to do this project in pairs (except for the extra credit), and are strongly encouraged to do so.

This project is due on Friday, December 13th, at 11pm.
Late submissions for this project will not be accepted.

Designing Your Program

Before you start writing the code for your program, you need to plan out what classes, functions, and variables you'll need in order to implement the game Snake. You have been provided with a template file, `project10_template.txt`, that includes part of the design for you.

The design for the class `SnakeGame` is completely done for you, as well as the descriptions (and definitions) of two helper functions, also described below. You will need to decide how to design the classes `ScoreBoard`, `Snake`, and `Apple`.

- `create_segment`. This function takes a color (given as a string) and a position, and creates and returns a Turtle object at the given position. This Turtle object is displayed as a circle with the given color. This is used to create the front and body of the snake, as well as the apple.

- `define_keys`. This function takes a `SnakeGame` object and a `Snake` object, and binds the keys to functions. This makes it so that you can control the snake with the arrow keys, and start a game by hitting return.

- `draw_boundary`. This function draws a rectangle showing the edges of the snake window, so that we can see the boundary for the snake.

You should start by reading through the description of the `SnakeGame` class. Use the design for this class to determine what functionality you need for the `ScoreBoard`, `Snake`, and `Apple` classes. Here is some information about each of the classes, you will need to figure out what member variables and member functions each of them requires.

- `Apple`. This class is used to represent the red apple that the snake is trying to eat. This class should have one member variable - what should it be? (Hint: use a helper function to initialize this member variable.) Then, looking at what is needed for the `SnakeGame` member function `play`, what member functions should this class have?

- `Snake`. This class is used to represent the snake that we move around the window. The member variables for this class are provided for you, as well as the constructor and the member functions that are bound to the arrow keys (so the player can change the direction of the snake). A snake is represented by a `front`, which is a Turtle object created by the `create_segment` function (in the color black). It also consists

of a `body`, which is a list of Turtle objects created by the `create_segment` function (in the color grey). Finally, a snake object has a `direction`, given as a string, which is the direction that the snake is currently facing. Looking at what is needed for the `SnakeGame` member function `play`, what additional member functions should this class have?

- `ScoreBoard`. This should track the current score and high score, and has the ability to display these scores. The scores are displayed using a Turtle object called `pen`; what other member variables are required? Looking at what is needed for the `SnakeGame` member function `play`, what member functions should this class have?

Before you move on to writing the code for your program, check your design with your lab instructor, to verify that you're on the right track.

Writing Your Program

Now, you will start writing the actual code for your program, in a python file called `snake.py`. This file has been started for you, and is posted on moodle. As you define these classes, be sure to test every function that you write before moving on to the next one - don't try to write the entire program all at once, or you're likely to end up with a lot of errors that are very difficult to track down.

Here are some notes to help you as you implement your design.

- `Apple`
  - In `random`, there is a function `randint` that can be used to generate a random integer in a given range. For example, the function call `random.randint(0,100)` will return a random integer between 0 and 100, including 0 and 100.
  - You may find the turtle function `goto` useful here.
  - The width of the snake window is 600, and the height is 500.
  - Remember to write contracts and docstrings for all member functions.
- `Snake`
  - When you add a new segment to the body of the snake, the easiest way to do this is to place the new segment where the front of the snake currently is.
  - Moving the snake is tricky: you need to check which direction the snake is facing, and use that to decide how to move. You'll also need to figure out how to move the body of the snake. You may find the following turtle functions useful:
    * `xcor()`: returns the x-coordinate of the turtle.
    * `ycor()`: returns the y-coordinate of the turtle.

    Regardless of the direction, the snake should move 20 units, since that's the size of one of the segments.

  - You'll need to write contracts and docstrings for all member functions.

- `ScoreBoard`
  - This class is missing some member variables and member functions.
  - You'll need to write contracts and docstrings for all member functions.

- `SnakeGame`
  - The constructor for this class is defined for you, but you'll need to finish the definition of the function `play`.
  - In order to prevent your snake from moving too quickly, you will want to make use of the function `sleep` from `time`. For example, the call `time.sleep(.1)` will pause the program for 1/10th of a second, before resuming the program.

- You may also find the turtle functions `distance` and `position` helpful.

Note: if you're interested in programming your own games in python beyond this project, using cTurtle graphics is not a good choice. You might want to consider using the pygame library.

Extra Credit

Extra credit is to be completed individually.

For this project, you have the option to improve upon your snake program in a variety of ways, in order to earn extra credit. You may customize your game with any combination of the following, earning a **maximum of 6 total extra credit points**.

- Improve the aesthetics. Our game is functional, but not particularly visually appealing. This can be fixed with extra tinkering with the Turtle graphics, and should certainly include making the snake look more like a snake, and the apple look more like an apple. (1-4 points, depending on how impressive the improvements are)

- Choosing a level. Our game only has one level of difficulty, which might get boring for snake enthusiasts. Edit the program so that the player chooses a level of difficulty, which determines how quickly the snake moves. (3 points)

- Adding obstacles. Add randomly generated obstacles to the window. If the snake hits an obstacle, the player loses (the same as if the snake collided with itself or the boundary of the window). (3 points)

- Adding multiple apples. Instead of just one apple, there are now multiple apples in the window at once. (2 points)

- Different apple values. Instead of every apple being worth one point, each apple has a randomly generated value between 1 and 5. (1 point)

- Two players. Instead of having a single snake controlled by a single player, now there will be two snakes controlled by two players, competing to get the apples. The second snake should be controlled by the keys W,A,S,D. It is up to you to decide the rules for collisions. (6 points)

The customizations should **NOT** be made in your main submission, `snake.py`. Instead, they should be made in a copy of this file, called `extra_credit.py`.

If you have other ideas for ways to customize your snake game, ask Prof. Lynn, and she will assign a point value to your idea.

Submitting your work

For this project, you will need to submit the following:

- `project10_template.txt`, which is the completed text template file, containing your design for your snake program.
- `snake.py`, which is a Python file containing the implementation for your snake program.
- (optional) `extra_credit.py`, a Python file containing your customized snake program.

You will be submitting your files using Moodle. If you worked with a partner, include both of your names in the file `project10_template.txt`, and make only one submission for `project10_template.txt` and `snake.py`. The extra credit is to be completed and submitted individually.

Grading

You will earn one point for each of the following accomplishments:

- (4 points) You have submitted a correct design for your project, including lists of member variables for each class, and descriptions of the member functions.
- You have written correct contracts and docstrings for the member functions of `Apple`.
- You have written correct contracts and docstrings for the member functions of `Snake`.
- You have written correct contracts and docstrings for the member functions of `ScoreBoard`.
- When the game starts, the snake, apple, and scoreboard are all displayed.
- Your snake moves in its current direction.
- The body of the snake moves correctly following the front.
- The direction of your snake can be controlled with the arrow keys.

- When the snake collides with the edge of the window, the snake is reset.
- When the snake collides with itself, the snake is reset.
- When the snake is reset, the length of the body is reset.
- When the snake collides with the edge of the window or itself, the scoreboard is reset.
- When the snake finds the apple, the apple is moved to a new position.
- When the snake finds the apple, the score is increased by one.
- When the snake finds the apple, the body of the snake grows by one segment.
- The high score is updated and maintained correctly.
- Your code is well-organized, easy to understand, and not unnecessarily complicated. Any particularly tricky parts are explained with comments.