CS 172 – Lab 5: PyGame

A Winter Wonderland

In this lab, we'll use OOP for graphics programming and add some interactivity to the application.

Provided Code

You are provided with two files. Study each file before creating your own classes.

- **drawable .py:** that contains an abstract base class called `Drawable`. See details below.
- **main.py**: The main program. Here is where the main script will go. The file provided contains the basic template to create a Pygame program.

The Classes

The Drawable Class

In this week's lecture we discussed how we can used object-oriented programming concepts to make graphics applications more organized and easier to create. One of the things that we'll leverage is inheritance, polymorphism, and abstract base classes.

We have included `drawable.py` that contains an abstract base class called `Drawable`.

This class has the following methods:

- A constructor (`__init__`) method where you can set the `x` and `y` locations for your object.
- Accessor and mutator methods for the `x` and `y` locations.
- An **abstract** method called `draw` that takes a surface to draw on as a parameter.

Part 1 – Milestone 1: The `Rectangle` class

Now we're going to derive from the `Drawable` class. The first derived class you'll want to make is `Rectangle`.

To instantiate the `Rectangle` class you need:

- The (`x`, `y`) location where the `Rectangle` is to be drawn.
- It's `width` and `height`.
- It's `color`.

Then in the class's `draw()` method you draw a rectangle starting at `(x, y)` of dimensions `(width, height)` on the surface in the chosen `color`.

NOTE: For credit you **must** derive from `Drawable` and use its constructor.

Once you have implemented the `Rectangle` class, go to the **main.py** file. You will find hardcoded test for the `Rectangle` class. Run **main.py** in Thonny and see if these test pass. If so, this is a good point to submit your code and earn 20 points! You will get partial lab credit for completing this part, even if you do not complete the rest of the lab.

Part 2 – Milestone 2: The `Snowflake` class

Next up let's create a `Snowflake`!

A `Snowflake` will be made up of 4 lines. If the `Snowflake` is to be centered at `(x, y)` then those four lines are:

- Line1: `(x - 5, y)` to `(x + 5, y)`
- Line2: `(x, y - 5)` to `(x, y + 5)`
- Line3: `(x - 5, y - 5)` to `(x + 5, y + 5)`
- Line4: `(x - 5, y + 5)` to `(x + 5, y -5 )`

To instantiate the `Snowflake` class you need the (`x`) location where the `Snowflake` will start (its `y` location will always start as `0`)

Your `draw()` method draws the four lines as mentioned based on the current `(x, y)` location in `white`.

NOTE: For credit you **must** derive from `Drawable` and use its constructor.

Once you have implemented the `Snowflake` class, go to the **main.py** file. You will find hardcoded test for the `Snowflake` class. Run **main.py** in Thonny and see if these test pass. If so, this is a good point to submit your code and earn 20 more points! You will get partial lab credit for completing this part, even if you do not complete the rest of the lab.

The Main Application

Part 3 – Milestone 3: the basic animation

Here's the main application:

- Draw a "ground plane". Presumably a green rectangle
- Draw a "sky plane". Presumably a blue rectangle

- In every iteration of your graphics loop
    - Loop through all your `Drawable` objects and draw them.
    - If the current `Drawable` is of type `Snowflake` (**hint**: use the `isinstance` function) increment the `y` coordinate of that `Snowflake`.
    - Spawn a new `Snowflake` instance (adding it to your list of `Drawables`) at a random `x` location (with `y` initially set to `0`).

Below is a sample image:

If you have this part working, this is a good point to submit your code and earn 35 more points! You will get partial lab credit for completing this part, even if you do not complete the rest of the lab.

Part 4 – Milestone 4: Interactivity

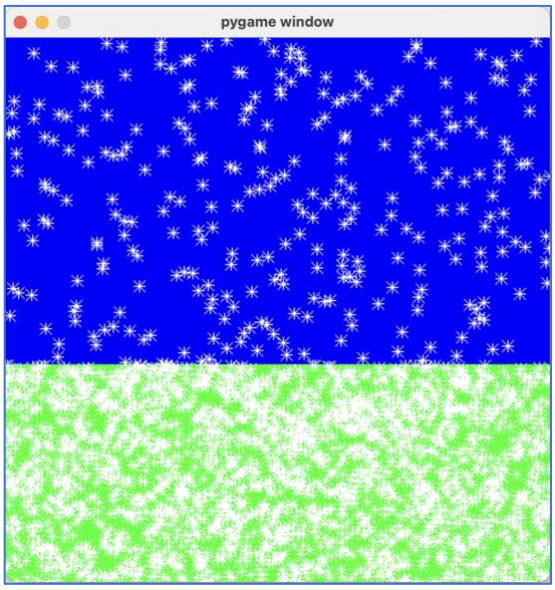
Finally, we'll allow the spacebar to toggle the animation. That is, the first time it is hit, all the snowflakes stop falling. The next time, they resume falling. etc.

If you have this part working, this is a good point to submit your code and earn 15 more points! You will get partial lab credit for completing this part, even if you do not complete the rest of the lab.

Extra Credit (10 additional points)

For extra credit, when a `Snowflake` is spawned also assign it a maximum `y` value that is somewhere (randomly) between the start of the ground plane and the bottom of it. Then when a `Snowflake` hits its maximum `y` value, it no longer animates. This should make it look like the snow is sticking to the ground!

Below is a sample image:



Scoring

The score for the assignment is determined as follows:

- 20 points - Correctly implemented `Rectangle` derived class.
- 20 points - Correctly implemented `Snowflake` derived class.
- 35 points - Create the basic animation correctly.
- 15 points - Spacebar correctly toggles animation.
- 10 points - Program style: attribute mangling, good variable names, comments.
- (10 pts EC) - Snow "sticks" to the ground plane.

Note: Please make sure you put write both lab partners' names and userIDs (in the form abc123) in the header comment of the file you submit for this assignment.