CS 172 – Homework 3

Purpose:

After completing this assignment, you will have practiced using the Pygame graphics API to create a fun 2D interactive game! Graphics are a great application of object-oriented programming.

Description

For this assignment you will write a simple game where the object is to hit a stack of blocks with a ball, and knock all the blocks in the stack.

Specification for the classes:

The Drawable class

In 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.

Your first task will be to create your own version of the `Drawable` abstract base class (similar to the one we used in class last week). This class should have the following attributes:

- `position`: `x` and `y` location for the center of the object
- `visible`: - A Boolean variable (`True` or `False`) such that if `True` the object is draw, if `False` it is not. Although you may not choose to use this attribute, it could be useful.

and the following abstract methods:

- `draw`: Takes as a parameter a surface to draw on.
- `get_rect` - returns a Pygame `Rect` object that is the bounding rectangle that fits tightly around your object.

Feel free to implement constructors, inspector and mutator methods, along with addition attributes and methods as you see fit.

Put all this code in a file name **`Drawable.py`**

The Ball class

The `Ball` class inherits from `Drawable` and it will draw a circle at its current location. You must implement at the very least the required methods of the base class (`draw` and `get_rect`), as well as a constructor. You may need to implement other methods as part of the public interface.

Put all this code in a file name **`Ball.py`**

Block class

The `Block` class inherits from `Drawable` and it will draw a square with a black outline at its current location. You must implement at the very least the required methods of the base class (`draw` and `get_rect`), as well as a constructor. You may need to implement other methods as part of the public interface.

Put all this code in a file name **`Block.py`**

The Text class

The `Text` class inherits from `Drawable` and it will be used to display the player's score. You must implement at the very least the required methods of the base class (`draw` and `get_rect`), as well as a constructor. You may need to implement other methods as part of the public interface.

Put all this code in a file name **`Text.py`**

The main Script

Set up the scene

In the file **`hw3.py`** initialize Pygame and create a window. In this window put a black line (a "ground plane") across the window. It might help to make this line a `Drawable` derived class. Next, add an instance of the `Ball` class at a starting location of your choice. Finally, create at least 6 instances of the `Block` class, placed throughout the scene. They are the targets for the ball. In addition, place a `Text` object at the top of the window to display the score. See figure 1:

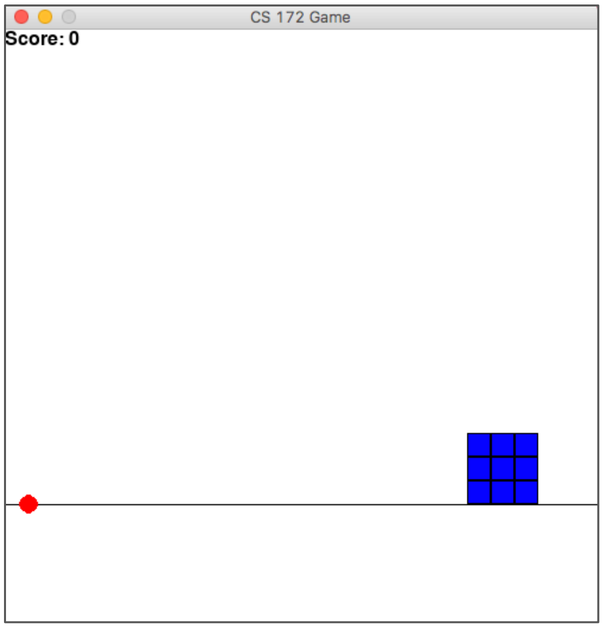

Figure 1: Initial Scene

The animation: Launching the Ball

We want to launch our ball by dragging the mouse. The difference in locations between where you press the mouse down and release it will determine the velocity and angle of the initial trajectory of the ball. We will use some physics to determine the velocity and angle.

Add to your main loop handling for `MOUSEBUTTONUP` and `MOUSEBUTTONDOWN` events. When `MOUSEBUTTONDOWN` or `MOUSEBUTTONUP` occurs:

- Store the respective mouse location
- On `MOUSEBUTTONUP`
  - Compute the initial x-velocity, `xv`, as the difference in the x-coordinates when the mouse was depressed vs released.
  - Compute the initial y-velocity, `yv`, as negative one times the difference in the y-coordinates when the mouse was depressed vs released. Remember: The positive y-axis points down in Pygame.

After launching the ball (which will initialize `xv` and `yv)`, as long as `|yv| < 0.0001` you should update the ball's position at the end of your game loop. But first we must define a few constants:

- Create a variable, `dt`, to store the "delta time". You can play with this parameter, but as default set it as `0.1`
- Create a variable `g`, for gravity. Set it to `6.67`
- Create a variable R, which is the "rebound" constant. Set it to `0.7`
- Create a variable `eta`, which is the coefficient of friction constant. Set it to `0.5`

Now, our update rules for the ball are as follows:

- The ball's x-coordinate should increase by `dt * xv`
- The ball's y-coordinate should decrease by `dt * yv`
- If the ball is below the ground (`y > 400`), we want to reverse its y-velocity by some amount and reduce the x-velocity (to account for friction). In particular `yv = -R * yv` and `xv = eta * xv`
- Otherwise (if the ball is above the ground), set `yv = yv - g * dt`

NOTE: Since the formulas provided above use floating-point values, you will likely want to keep the ball's location as a floating point number. However, Pygame's `draw` methods require integer locations. So just cast your locations as integers when you pass them into the `draw` functions. For example:

```
pygame.draw.rect(..., ..., (int(x), int(y), 20, 20))
```

The animation: Hitting the Blocks

Finally let's hit those blocks! At each iteration of the game loop, loop through all the blocks and test to see which intersected with the ball. For any block that does, turn their `visibility` to `False` and update your code as necessary so that objects that have their `visibility` set to `False` aren't drawn. Also update the score in the `Text` object to reflect that the player has hit some blocks. Here is an example of what the game looks like when the user has hit a few blocks:

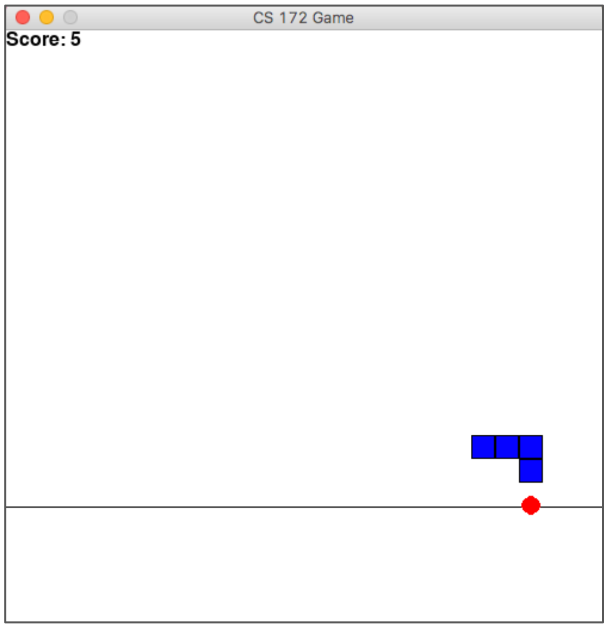

Figure 2: After hitting a few blocks

Below is code for a function that returns if two rectangles intersect:

```
def intersect(rect1, rect2) :
    if (rect1.x < rect2.x + rect2.width) and
        (rect1.x + rect1.width > rect2.x) and
        (rect1.y < rect2.y + rect2.height) and
        (rect1.height + rect1.y > rect2.y) :
          return True
    return False
```

If you have time, you can add other things to the game. Here are some ideas:

- Add a background image to provide an "environment"
- Add a "high score" component, where you store the high score in an external file to be read from and written to.
- Make your flying object more interesting than just a circle

NOTE: Please keep in mind that you are expected to write a good quality, well formatted program. That means:

- Your files must have a header comment listing your full name, Drexel user id, and the purpose of the file – at the very least.
- Repetitive code (code that appears in multiple places in the main script) should be written as a function.
- Your program must use good style, including proper identifier names, useful comments, and proper use of indentation and whitespace.
- You program should also have an appropriate user interface so that anyone one using the program knows what to do and what to expect.

Grading

Criteria	**Points**
`Drawable` abstract base class	10
Properly implement `Ball` derived class	10
Properly implement `Block` derived class	10
Properly implement `Text` derived class	10
Main script: set up of initial environment	15
Main script: `Ball` interaction/animation (physics)	10
Main script: `Block` animation/collision detection	10
Main script: score updated properly	10
Code follows good style guidelines: good variable names, comments, proper use of white space, and separate repetitive code into functions	10
All files have a header comment with your name and user id	5
Total possible points	**100**

NOTE: If you code has any runtime errors a 50-point deduction will be taken. Only portions of the code that execute without errors will be graded. If your script cannot run at all, you will receive 0 points.

How to Submit your assignment:

- Assignments must be submitted via Blackboard Learn.
  - Please note that assignments submitted via email will not be accepted.
  - Late assignments will be penalized by 1% off per hour, up to 48hrs (after which they will not be accepted)

- For this assignment, you must submit a single zip (such as **`HW3.zip`**) file that contains:
  - **`drawable.py`** – file that contains the drawable class.
  - **`block.py`** - file that contains the block class
  - **`ball.py`** - file that contains the ball class.
  - **`text.py`** - file that contains the text class.
  - **`hw3.py`**- your main script (the animation)

Academic Honesty

You must be the **sole original author** of the **entire solution** you submit. You must compose all program and written material yourself. All material taken from outside sources (e.g. textbooks, in class examples, labs, etc.) must be appropriately cited.