Simple Python “Games” pt. 2

Contents

1 Skills/Concepts

This module is a continuation of the Python Games module.

This module will continue to build on the previous concepts of:

- SDL
- blitting
- Drawing APIs

In addition, this module will introduce classes and inheritance.

2 Preparation

Continue with the same environment from the previous (Python Games) module.

We will be using an existing git repository that contains a fair amount of code instead of producing a new program from scratch. This is to avoid copy and paste errors and allow us to cover material faster.

To get the code, execute the following command:

```
git clone https://www.github.com/cyberpost500/modules/
```

Then extract the project repository to your desktop:

```
cd modules/python-games/
tar -C ~/Desktop/ -xf project.tar
```

The code will be in the Desktop/project directory.

3 Shapes

3.1 Background

In addition to blitting text and filling a region of pixels with a color, simple shapes are also possible. Simple shapes will help to form a basis for more complicated programming structures.

These APIs are demonstrated on the `cirlces` branch of the project repository. Switch to that branch with the following command:

```
git checkout circles
```

3.2 Review/Run/Modify the Code

0. View the Documentation for the program: ~~ pydoc game _ _ ~~

Press *Q* to exit when you're done.

1. Run the Program:

```
python game.py
```

2. Move the position where the circle is drawn:

```
pygame.draw.circle(screen, (0xff, 0xff, 0), (shape_x, shape_y), 10, 1
```

This is the line responsible for drawing the circle. The position where the circle is drawn is controlled by variables `shape_x` and `shape_y`.

Modify either the values passed on line 58 or change the initial values for `shape_x` and `shape_y` on line 43.

3. Save and run your program.
4. Add another shape to screen as well. Some other functions:

Function	Description
pygame.draw.rect	rectdraw a rectangle shape
pygame.draw.polygon	polygondraw a shape with any number of sides
pygame.draw.circle	circledraw a circle around a point
pygame.draw.ellipse	ellipsedraw a round shape inside a rectangle
pygame.draw.arc	arcdraw a partial section of an ellipse
pygame.draw.line	linedraw a straight line segment

Look up help for the function you want with pydoc:

```
pydoc pygame.draw.circle
```

Add the instruction for drawing the new shape directly beneath the instruction for drawing the existing circle.

5. Save and run your program.
6. Revert the file changes back.

Undo your edits to the file using the following git command:

```
git checkout game.py
```

3.3 Conclusion

Now the program displays a circle and optionally any other shapes you may have put into it. The arguments used to make the circle are similar to those used for the other primitive shapes, there shouldn't be anything too surprising. But, if the Python documentation is a little daunting, feel free to search online for other examples.

4 Animated Shapes

4.1 Background

Animated shapes via primitive graphics is much like the old fashioned cartoon animations. A new image is drawn that is slightly different than the last, providing the illusion that the object has moved from one location to another. Accomplishing this with Python is as simple as changing the coordinates used to draw successive versions of the shape being animated.

These APIs are demonstrated on the `animated_cirlces` branch of the project repository. Switch to that branch with the following command:

```
git checkout animated_circles
```

4.2 Review/Run/Modify the Code

0. View the Documentation for the program: ~~ pydoc game ~~

Press *Q* to exit when you're done.

1. Run the Program:

```
python game.py
```

2. Change the speed and/or heading of the animated circle.

Look at line 45 for the assignment of the "speed"

Look at line 31 and 32 for the modification of the x and y position of the shape.

Modify these lines at will, but predict what the outcome will be before running the program.

3. Save and run your program.
4. Revert the file changes back.

Undo your edits to the file using the following git command:

```
git checkout game.py
```

4.3 Conclusion

The shape is now moved around the screen at will, but once it moves off an edge, it never comes back. In fact, it appears as though the shape is continually moved off in that direction even though it can't be seen.

5 Bouncing Shapes

5.1 Background

The moving shape will continue off the screen, but it would be better to make it respond to the edge of the screen by "bouncing" but remaining on screen.

These APIs are demonstrated on the `animated_cirlces_2` branch of the project repository. Switch to that branch with the following command:

```
git checkout animated_circles_2
```

5.2 Review/Run/Modify the Code

1. Run the Program:

```
python game.py
```

2. Make it run faster.

Edit the speed so that it reaches a boundary faster.

3. Change the code that controls where the shape will bounce by replacing one of the array values with a new value or re-assigning the array at a given offset to a new value.

Line 48 assigns the screen size. The easiest way to modify this is to directly modify the array. For example:

```
bounds[1]=10
```

If placed below line 48, can you predict where the shape will bounce?

Another way to modify where the bounce occurs is to change one of the test statements on line 72, 75, 78, or 81 (and the following assignment).

4. Save and run your program.
5. Revert the file changes back.

```
git checkout game.py
```

5.3 Conclusion

The shape is now moved around the screen at will and bounces off an edge of the screen. These features work, but they're not obvious in how they work and this slows down the speed that someone can read, and therefore maintain, this code.

6 Shape Class

6.1 Background

Abstracting related functions into a library or a class can help to ease readability and improve reusability of a body of code. This means that it's potentially easier to re-use and easier to read and understand.

These APIs are demonstrated on the `shape_class` and `moving_shape_class` branches of the project repository. Switch to that branch with the following command:

```
git checkout shape_class
```

6.2 Review/Run/Modify the Code

0. View the Documentation for the program:

```
pydoc game
```

Press *Q* to exit when you're done.

1. Run the Program:

```
python game.py
```

2. Notice that the program content is a step backwards in functionality. The shape no longer moves, but take a moment to review the content of the Shape class.
3. Notice that the only uses of the Shape class in the main function are creation of the shape, and telling the shape to draw itself to the board.
4. Now switch to the `bouncing_shape_class` branch with the checkout command:

```
git checkout bouncing_shape_class
```

Take a moment to compare the content of the Shape and MovingShape classes.

5. Take a moment to cmopare the MovingShape and BouncingShape classes.
6. Add a new BouncingShape to the program by duplicating lines 142, 156, and 157, for example, the following on line 143:

```
bouncer = BouncingShape((42, 36, 90), 0, 0, 10, 1, 20, bounds)
```

And the following on line 157 and 158:

```
bouncer.move()
bouncer.draw()
```

7. Save and run your program.
8. Revert your changes back.

```
git checkout game.py
```

6.3 Conclusion

You should start to see how classes can make a program flow cleaner by removing all the nitty gritty details and instead the main just calls abstract functions to do the work that needs to be done.

7 New Classes

7.1 Background

Creating new classes from scratch provides a new functional object that accomplishes some specific tasks and acts as a basis for a new object that is can leverage the existing class but add some new functionality.

The code to be modified is in the `clear_bg` branch. It is essentially the same as the previous branch but with the addition that new shapes are drawn to a clean screen instead of drawing over top of each other.

```
git checkout clear_bg
```

7.2 Review/Run/Modify the Code

0. View the Documentation for the program:

```
pydoc game
```

Press *Q* to exit when you're done.

1. Run the Program:

```
python game.py
```

2. Take a moment to review the content of the file, just skimming through it is fine.
3. Create a new bouncing shape class for drawing a rectangle instead of a circle and add it to your program.

The new class will only need to define one method: draw.

```
class BouncingRectangle(BouncingShape):
    '''
    Creates a bouncing rectangle much like the BouncingShape, but uses a
    rectangle instead of a circle.
    '''
    def draw(self, surface):
        half = self.size/2
        top = self.x-half
        left = self.y-half
        rectangle = pygame.rect.Rect(self.x, self.y, self.size, self.size)
        pygame.draw.rect(surface, self.color, rectangle, self.border)
```

4. Create an instance of the new class and put it into the shapes list.

```
shapes.append(BouncingRectangle((0, 0xff, 0xff), 10, 0, 10, 1, 7, bounds))
```

You may want to use a differnt starting size or position so that it doesn't follow the previous shape around the screen. The value is left the same here for illustrative purposes.

5. Save and run your program.
6. Create another bouncing shape class, and make it show a smiley face or other composite shape.

7.3 Conclusion

This last modification should show how useful classes are for leveraging an existing class to accomplish some slightly different task or behavior.