Simple Python “Games” pt. 4

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
- Classes
- Loops
- Member Functions
- Finite State Machines

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 Menus and more States

3.1 Background

The menuing system has been updated to include animations that are very similar to the game play experience. This gives it a more polished look, but it would greatly benefit from having a new menu option that explains how to play the game for the user.

The code to be modified is in the `master` branch. This branch contains a fully functional game.

```
git checkout master
```

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
```

Use the arrow keys to navigate the menu.

Eventually choose the quit option from the menu.

2. Take a moment to review the content of the file, just skimming through it is fine.
3. Add a new menu option called help.
4. Make the function it calls something like `dohelp`.
5. Add a new game state called `HELP_STATE`, similar to the existing states, and have the `dohelp` function set this state, similar to the `reset` function.
6. Add an if test to the main function that looks for the `HELP_STATE` and displays text explaining the game for the user. You can model it after the `WIN_STATE` or `LOSE_STATE`.
7. Add an if test to the main function near the other lines controlling input handling and have the test put the state back to the menu state when the user presses the escape key. Again, you can model it after the `WIN_STATE` or `LOSE_STATE`.
8. Save and run your program.
9. Commit your changes to the code. For example:

```
git commit game.py -m 'Added a help option to the menu'
```

3.3 Conclusion

These modifications should have helped to make you familiar with the new program flow and re-expose you to callbacks (the `dohelp` function), global variables (the STATE related variables,) and finite state machines and their role in the overall game.

4 Playerier Player

4.1 Background

The player shape was recently added to provide the user a shape that they can move around the screen. It's a red circle with the letter P in the center. It's a nice place holder, but it would look better with some customizations.

4.2 Review/Run/Modify the Code

1. Confirm that the code is still working by launching the game and executing each menu option before continuing.

```
python game.py
```

2. Locate the `PlayerShape` class. Skim through the `draw` function and see what it does.
3. Move to the end of the `PlayerShape` class definition and add a new `SmileyPlayerShape` class.
4. Add a new draw function to this class, you can copy and paste the draw function from the player class as a starting point.
5. Instead of drawing a letter P, make a smiley face by creating two additional circles and an arc.

Use the following functions:

```
pygame.draw.circle(Surface, color, pos, radius, width=0) -> Rect
arc(Surface, color, Rect, start_angle, stop_angle, width=1) -> Rect
```

Documentation for arc:

> Draws an elliptical arc on the Surface. The rect argument is the area that the ellipse will fill. The two angle arguments are the initial and final angle in radians, with the zero on the right. The width argument is the thickness to draw the outer edge.

Retrieved from here

6. Modify the `initialize` function so that it uses the new class instead of the PlayerShape class.
7. Save and run your code.
8. Commit your changes to the code. For example:

```
git commit game.py -m 'Improved the player shape by adding a better one'
```

4.3 Conclusion

This module served to help re-expose you to extending classes and making use of library functions.

5 Customize Customize Customize

Make further customizations to the code.

1. Come up with an objective.

For example:

- Adding a score to the game based on the time spent playing and the number of shapes consumed.

- Modify the triangle shape (by changing the base `Shape` and the `MovingPolygon`) so that it can resize when it eats other shapes. Moving the resize logic out of the `remove_and_resize` function and putting it in the base `Shape` will make it easier to change the size.
- Add sound effects when eating other shapes, when winning, or when losing.
- Add music to the game play.
- Add logic that causes the NPC shapes to try to eat food items and follow the player instead of moving randomly.
- Or anything else you might want to do.

2. Scope and plan the approach to the problem to see if you can already do it with what you know, or if you will need to learn more. Research the elements you think you might need to see if you will be able to learn enough to complete the objective during the post meeting.

Don’t immediately discount something as too hard or taking too long.

Chances are that you can find code snippets if you Google your topic and look for Stack Overflow posts.

3. Ask questions and run ideas by the advisers, we can probably give you some time saving pointers
4. Don’t forget to save before running.

5.1 Conclusion

This module should help to make you more comfortable with learning other people’s code and modifying it to do what you want. In addition, it exposes you to the sorts of concepts you might consider during actual design or implementation of an idea.