pyGame Lecture #7

(Examples: dotgame with menu)

MULTIPLE SCREENS IN PYGAME

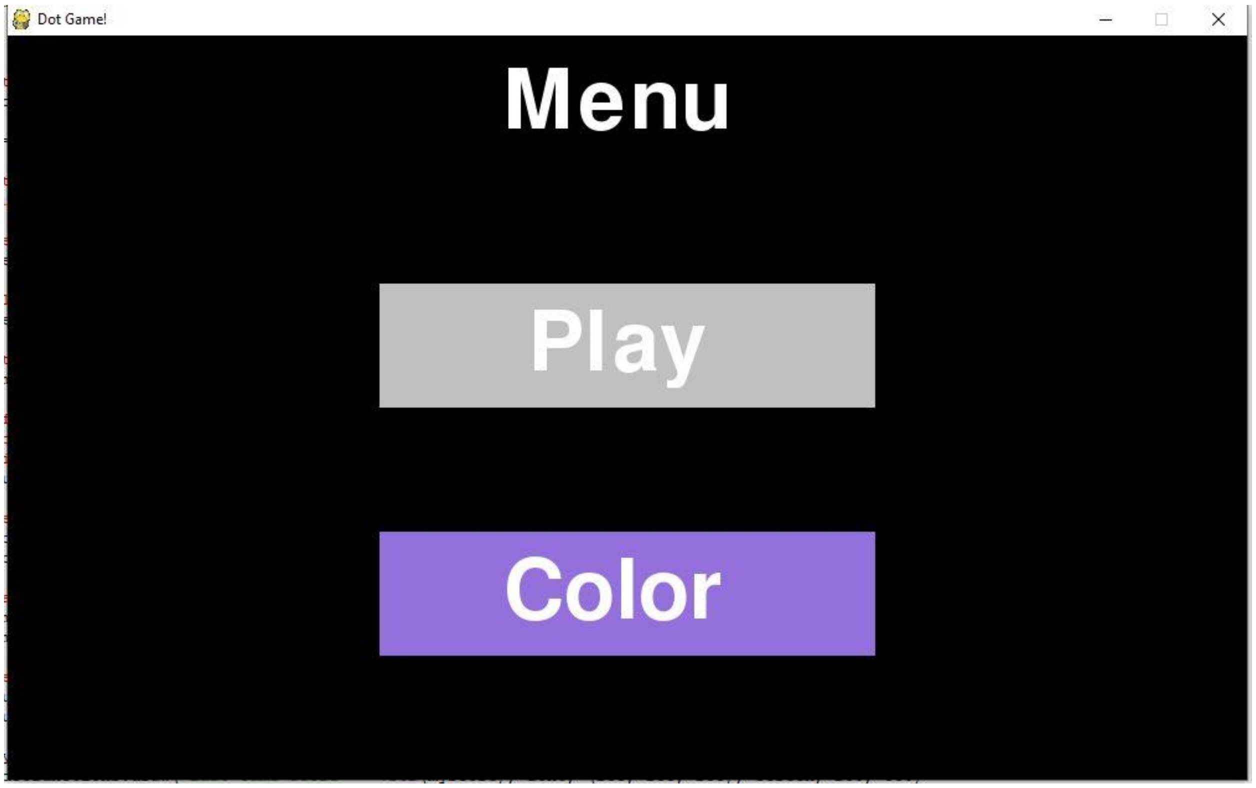
I. What is the use of multiple screens?

When a game "finishes", it would be nice to see the results or get a chance to play the game again. In order to do this smoothly, the display should essentially look like a different screen. Technically, in this lecture, we won't open separate windows, but we'll learn how to essentially "toggle" between two screens. To keep things simple, we'll use the same exact Dot Game example with which you are already familiar, so you can just focus on the changes from the multiple file version of the code that allow us to use different screens and play the game multiple times.

II. The Mechanics of Using Different Screens

We will have one main function, which will instantiate the screen and offer the menu screen, so to speak. This menu screen will have some buttons to press. On the event that a button is pressed, then we can call a function and that function can "play the game". Instead of that function only quitting when the user exits, we can have that function complete and return some information, maybe the score of the game, when it is done.

In the dot game example, our initial screen looks like this:



The "buttons" are simply rectangles on the screen. There's a method called "collidepoint" that can be called on rectangles to detect if a particular point is within the rectangle or not. We can use the mouse.get_pos() function to retrieve the position of the mouse at the time of a click.

Thus, if we look at the code in dotgame.py, in the MOUSEBUTTONDOWN code, we can see that we if we hit either of the buttons, either the

`main_game or options` functions from the dotfunctions.py file gets called.

In both instances, those functions eventually return a value to function menu, and when this occurs, the menu screen gets drawn again as per the directions in THIS while True loop. (So, THIS while True loop, runs really fast until one of these two functions gets called, and then it just sits there while a different while True loop runs and eventually completes.)

III. Each Individual Screen

First, let's examine the options() function. This "program" allows the user to choose the color of the dots to "eat." It's set up much like one of the other example programs. It has a game loop, but eventually, instead of running until someone hits the quit button at the top left, there's another way for this function to stop running, and that's to return a value to the function that called it.

In the options() function, instead of being a while True loop, the loop is written as:

```
while running:
```

Running is initially set as True, but there are ways in which it can be changed to false in the game loop. In particular, whenever a clicks the mouse in one of the three color button locations, the variable color is set and running is set to False. This will stop the loop, and the one and only line after the loop returns color to the function menu.

The only other thing to note in this function is the call to the function draw. The function draw is a nice utility function that allows you to draw some text using a particular font and color to a particular location on the screen. This is a great example of a reusable function that aids the modularity of a program.

The more complicated function is the one that runs the dot game. This basically has a lot of the code from the main part of the original dot game. Before the game loop, all the setup code is included. Nearly all of the code in the game loop is identical, **EXCEPT** how the game ends. In the old version, if the player died, then the game just quit and printed the score on the console. In this version, we detect the death of the player via a Boolean variable alive, and when the game loop detects that alive is false, it immediately returns the player's score to the menu function.

Alternatively it's possible to end a game (and its corresponding screen) without returning anything, but you do have to exit the game loop. At that point, the code should finish running and return control to the function that called it.