pyGame Lecture

(Example: Fruit Game)

MULTIPLE SCREENS IN PYGAME

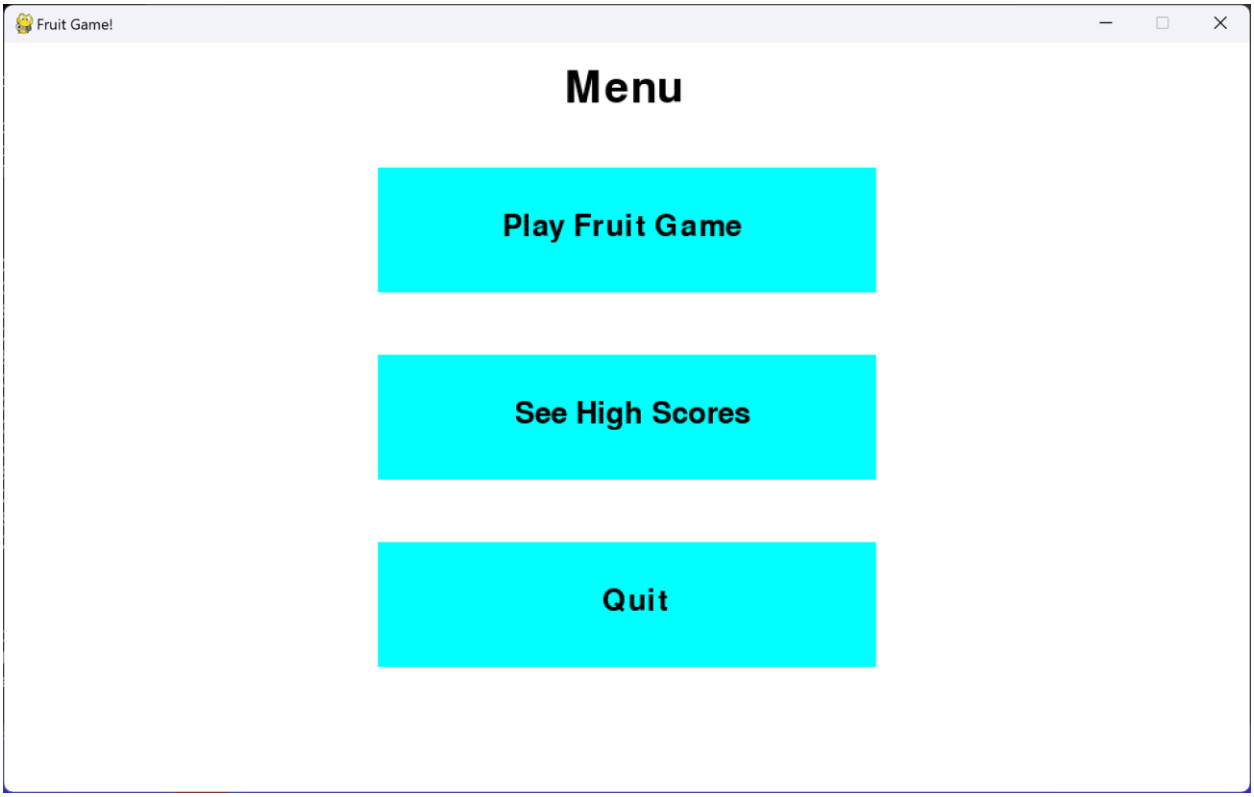
I. What is the use of multiple screens?

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. We'll look at one example for this lecture:

1. Fruit Game with a menu where the user can look at play, look at high scores, or quit.

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. (Or it might just be a void function that just executes a return statement.



The "buttons" are simply rectangles on the screen. We'll use the built in Pygame method collidepoint along side with getting the mouse position on a mouse click to detect if the user has made a menu selection.

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

`maingame` function from fruitgame.py or the `show` functions from highscores.py gets called.

In both instances, those functions eventually will execute a return statement (if used appropriately), 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

Screen 1: Main Menu Screen

First, let's examine the menu() function in the file fruitgame.py. This "program" allows the user to play the game, view the high scores, or quit. It's set up much like one of the other example programs. It has a game loop, and draws the screen, but when one of the mouse buttons is clicked, it will make the functions calls previously described:

```
fruitgame.maingame()
highscores.show()
```

In addition, this function has multiple options to quit: the usual x at the top right corner, the escape key, or the actual quit button. The loop here is our usual while True loop since this is our main function.

Screen 2: Fruit Game Screen

When the function maingame() gets called in the fruitgame file, it runs like normal doing lots of set up, asking the user for their name, and loading the high scores. The game loop is the same with "while True". The game in this function has two modes: playing or losing. A boolean variable lose keeps track of which of the two modes the user is in. The variable lose can get set to True (its initial value is False) by the player either hitting a bomb or dropping 20 fruits. Once lose is set to true, the high scores are updated and then game mechanics give two seconds until an if statement is triggered that simply executes the return statement, ending the function. Here is that whole set of processing code that occurs after each display update in the game loop:

```
# Lose when you drop 20 or more.
if dropped >= 20:
    lose = True

# Move the drops for the next iteration and remove useless ones.
if not lose:
    move(fruit)
    move(bombs)
    dropped += removeUseless(fruit)

# We've lost - update score and set lose flag.
if lose and loseT == 0:
    scorelist = update("highscores.txt",scorelist, name, score)
    loseT = time.time()

# Return to main screen.
if lose and time.time()-loseT > 2:
    return
```

So the key is that instead of quitting when the game is over, we simply make this function terminate via the return statement. (If there was some information for this function to return to the calling function, that information could be returned. For example, an alternate design would have returned the user's name and score to the calling function and let the calling function store the new high score.)

Screen 3: High Scores Screen

This function is much shorter. It has the necessary imports and set up (it needs fruitgame because it calls a function there.) There's only one function in this file, show. The basic game loop is set up. Before the loop the scores are loaded from the file and a button is set up to return to the main menu. In the game loop, if we detect a button press we will simply return. Here is the relevant code from the event check:

```
# Looking for mouse button press.
if event.type == MOUSEBUTTONDOWN:

    # Just look at left mouse button.
    if pygame.mouse.get_pressed()[0]:

        pos = pygame.mouse.get_pos()

        # We just finish this function and go to the main menu.
        if button.collidepoint(pos):
            return
```

Otherwise, all we will do is draw the high scores and button to the screen:

```
# Set up screen, button and button text.
DISPLAYSURF.fill("white")
pygame.draw.rect(DISPLAYSURF, "green", button)
fruitgame.draw("Return to Main Menu", font, "black", DISPLAYSURF, SCORE_W//2-
135, SCORE_H-70)

# Display high scores.
for i in range(len(scorelist)):
    fruitgame.draw(str(i+1)+". "+scorelist[i][1], font, "blue", DISPLAYSURF,
150,25+45*i)
    fruitgame.draw(str(scorelist[i][0]), font, "blue", DISPLAYSURF,
400,25+45*i)

pygame.display.update()
clock.tick(30)
```

Note: Hard-coding coordinates is never a good idea for good game design, but when learning, it's sometimes easier.