pyGame Lecture

MULTIFILE OBJECT ORIENTED PROGRAMS IN PYGAME

(Example: fruitgameobj)

I. Why Do We Need a Program Split Among Multiple Files?

It's extremely difficult for a programmer to keep track in her head all of the variables and functions in a file that is thousands of lines long. In order for humans to reliably maintain and upgrade software, it's necessary that the person editing a piece of code only have to keep track of a relatively small number of things all at once. (Psychologists have shown that most humans can only keep track of 3-7 things in their short-term memory at once. This is why most of us can manage to remember a few phone numbers, but the average person has some difficulty remembering their credit card numbers.) Typically, in a python program, when editing a single line of code, in theory any other line of code in the same file may directly or indirectly affect that line. Thus, when tracking down errors, one typically might have to search all around a particular file.

One innovation that helped reduce this complexity was functions. When you are writing code ***inside*** of a function, if you declare a variable inside of the function, it only exists within that function and won't conflict with a variable outside of the function. In addition to any variables you may declare in a function, you can only use the formal parameters (the items written in the parentheses at the top of the function) and any global variables (defined in the file not within any function) in that function. This helps reduce the total number of things you must keep track of, especially if one limits the number of global variables.

However, after a while, even writing a program wiith many functions in one file gets tedious. Imagine having to scroll from line 1327 to line 4434 to track down a single function call!!! Or having to remember where a particular function was written in a file when the file has a hundred functions!!! It might take 10-15 minutes just to track down ***where*** physically in the code the offending function might be.

Thus, the next level of abstraction that helps the programmer deal with complexity in programs is splitting a program into multiple files.

The idea is as follows: in each file, you keep a small number of related functions and constants. In one file you have your "main" program, which you actually run. From that file, you directly or indirectly call all of the other functions, some of which may reside in this main file and others which may reside in other files. If you split up your files in a logical way, then based on what a function does, you'll quickly know which file it's defined in. Once there, since the actual file is pretty small, you should be able to find it quickly. This is the same idea behind organizing a bookshelf by placing all books on the same subject on one shelf. You can think of each file in your program as a "subject" and within that file you have multiple functions that fit within that subject. In the main file, you call various functions that may be located in different files. The organization by files allows for the programmer to focus on a smaller portion of code when editing. It also allows for multiple programmers to work on the same code base in parallel. It's this latter advantage that truly makes software more scalable.

II. The Mechanics of Splitting a Program into Multiple Files

The actual mechanics of splitting a program into multiple files isn't too difficult. With object oriented code, we typically want to place each different class in its own file. (With non-object oriented code you would group together related variables and functions and place them in a separate file.) In that file, import anything you need (pygame, etc.) for the functions in that file. Now, if you want to call functions from this file from a different file, first put an import statement in the second file. For example, if the file fruitgame.py makes calls to methods in PicItemFile.py, then at the top of fruitgame.py we write:

```
from PicItemFile import picitem
```

Then, when we want to call a method in PicItemFile.py we write the function call as follows:

```
objectname.methodname(parameters)
```

An example is when we detect whether or not a mouse click has hit one of the fruit objects (picitem object really). Here is the section of code that takes care of checking a single mouse click with each fruit and if it's a hit removes the fruit and adds points to the player:

```
for f in fruit:
    if f.hit(event.pos):
        score += f.pts
        fruit.remove(f)
```

Alternatively, if we have a file that isn't storing a class and just has different variables and functions in it then we would do the following import if the file was called fruitfunc.py:

```
import fruitfunc
```

and we would call a function stored in that file as follows:

```
fruitfunc.move(fruit)
```

Python allows for a programmer to mix both object-oriented and non object-oriented styles. (C++ is another language that allows for the mixing of styles. Java requires object-oriented design.)

III. Example - Fruit Game

For Fruit Game, we want to create a class to store a fruit and put that class in a different file. We can create a new class, adapting the ideas in the token class. Our new class will be called pictoken. This class will also store a picture and an integer number of points (if it's clicked/eaten) in addition to a defining rectangle, dx and dy values for movement. Let's take a look at this class in its entirety:

This class is similar to token, but manages an image instead of a

rectangle

```
import random
import math
import time
import pygame, sys
from pygame.locals import *

# Token Class we'll use for drawing objects in pyGame
class picitem:

    # Default settings.
    dx = 0
    dy = 0
    pic = None
    rec = None
    color = pygame.Color(0,0,255)
    pts = 0

    # Similar to token constructor with picture and pts set up.
    def __init__(self,myx,myy,mydx,mydy,mypic,mypts):
        self.dx = mydx
        self.dy = mydy
        self.rec = pygame.Rect(myx, myy, mypic.get_width(), mypic.get_height())
        self.pic = mypic
        self.pts = mypts

    # Call each frame.
    def move(self):
        self.rec.x += self.dx
        self.rec.y += self.dy

    # Returns true iff the grid coordinate mypos is within the picture
    # box of self. Pygame does this for us, so call Pygame's method!
    def hit(self, mypos):
        return self.rec.collidepoint(mypos)

    # Draws the picture associated with this object at its cur. Position.
    def draw(self, DISPLAYSURF):
        DISPLAYSURF.blit(self.pic,(self.rec.x, self.rec.y))
```

We'll have three functions are not part of any class which we will just place in fruitgame.py. (If there were quite a few more we might place these in a separate file.) Here are those method signatures:

```
# Moves each item in the list items by one "frame"
def move(items)

# Removes all items from the list items that have gone off the screen.
def removeUseless(items)

# Draws text in the given font and color on surface on surface's
# coordinates (x, y).
def draw(text, font, color, surface, x, y)
```

After these functions, we define our main function, where do quite a bit of set up (load images, set up points for fruits, create our main window, etc.)

In our game loop, the only event we look for is a mouse click hitting a fruit. Here is that code in its entirety:

```
for event in pygame.event.get():
    if event.type == QUIT:
        pygame.quit()
        sys.exit()

    # Looking to see if you tried to get a fruit!
    if event.type == MOUSEBUTTONDOWN:

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

            # Now see which fruit we hit! (I've implemented it so you
            # could hit more than one in a single click!)
            for f in fruit:
                if f.hit(event.pos):
                    score += f.pts
                    fruit.remove(f)
```

As it can be seen, the call to the method hit and accessing the instance variable pts for a picitem object works seamlessly.

Here is the segment of code where we create a random fruit once every 10 frames:

```
if step%10 == 0:
        x = random.randint(1, SCREEN_W)
        which = random.randint(0,3)
        mydx = random.randint(-2, 2)
        mydy = random.randint(3, 8)
        temp = picitem(x,0,mydx,mydy,pics[which],pts[which])
        fruit.append(temp)
```

Again, notice how we call the picitem constructor and it looks no different than if everything were in one file.

Finally all the drawing is easy in main since there is a draw method in picitem and a draw function for the text:

```
    DISPLAYSURF.fill(WHITE)

    # blit allows us to draw a surface onto another surface.
    for item in fruit:
        item.draw(DISPLAYSURF)

    # Display Score
    draw("Score: "+str(score), font, "black", DISPLAYSURF, 0, 0)

    pygame.display.update()
```

So to recap, if we want to split a program into multiple files, if the file stores a single class, in the file we want to use it, we write

```
from FILENAME import CLASSNAME
```

then, in the file the object is being used, just call each method normally.

If a file stores regular functions and variables, then do this import:

```
import FILENAME
```

Then to refer to any variables or functions do:

```
FILENAME.variable
FILENAME.functioncall(parameters)
```