pyGame Lecture #5

(Examples: fruitgame)

MOUSE INPUT IN PYGAME

I. Detecting Mouse Input in pyGame

In addition to waiting for a keyboard event to precipitate some action, pyGame allows us to wait for a mouse event. There are several different mouse events that can happen, but the most common one is a left button click on a pixel. This will be the only one covered in detail in these notes. For a full reference of mouse events, go to:

https://www.pygame.org/docs/ref/mouse.html

First, the event type we desire to look for is MOUSEBUTTONDOWN. Once we detect this type of event, to get more information, the key function we will call to see if a mouse button was pressed is:

```
pygame.mouse.get_pressed()
```

This function returns a list of boolean values of size 3. Index 0 stores True iff the left mouse botton was pressed, index 2 stores True iff the the right mouse button was pressed and index 1 stores True iff the middle mouse button was pressed. (Note that iff is not a typo. It is short for "if and only if". This means that if the corresponding mouse button was not pressed, that variable will store False.)

If we want to check to see if the left mouse button was pressed, we can do it as follows:

```
while True:

    for event in pygame.event.get():
        if event.type == QUIT:
            pygame.quit()
            sys.exit()
        if event.type == MOUSEBUTTONDOWN:
            if pygame.mouse.get_pressed()[0]:
```

Basically, once we know we have a MOUSEBUTTONDOWN event, if we call the pygame.mouse.get_pressed() function, we will get a list. Indexing this list at 0 will tell us if the left mouse button was the one pressed or not.

In addition to determining whether or not there was a left mouse button press, we may care to know which pixel value was clicked with the left mouse button press. (In most instances, we only perform an action if the mouse click occured on an icon, for example. It would be fairly crazy if some action occurred on an unrelated item to where the mouse click was made!) To extract this information, we simply use the expression:

```
event.pos
```

This returns a pair that stores the x and y coordinates, respectively, in pixels, of the event in question. We can use these values as necessary.

II. Fruit Game Example

A popular game for the iPad was Fruit Ninja. In the game, various fruits would fly on the screen and if you touched them with your finger, the fruits would disappear and you'd get points for each fruit vanquished. For this example, we'll have the user eliminate fruits via a mouse click. The fruits will appear on the screen once every 10 frames and fall from the top but be moving in random directions down. There will be four different types of fruits (apple, cherry, kiwi, strawberry) and each will be worth a different number of points. But each fruit of the same kind is worth the same. The game will end when you have let 20 fruits fall through to the bottom of the screen.

The fruits will be kept in a list, just like we kept all of the raindrops in a list in a previous example. We will retain the two important functions from the rain example: move and removeUseless as these are two general tasks that are also required in this program. Our removeUseless function will have a change compared to our previous version of it: it will return the number of items removed. This will help us keep track of how many fruits the user failed to vanquish. We'll add a function that takes in the location of a mouse click and the location of a fruit and determines if the fruit was hit or not.

III. Storing a single fruit

We will use a list of ***five*** items to store a single fruit: x coordinate, y coordinate, dx value, dy value ***and*** which fruit it is. The last item will be an integer, either 0, 1, 2 or 3. 0 will represent an apple, 1 will represent a strawberry, 2 will represent a kiwi and 3 will represent a cherry. This designation is completely arbitrary and made up by the programmer. In fact, any designation will do for the purposes of this program. The important thing is that the programmer uses ***the same designation*** throughout the program! This numerical mapping to the fruits is very useful because we can then store information about all the fruits in separate lists. For example, all the point values are stored as follows for our program:

```
pts = [50, 75, 100, 150]
```

This indicates that apples are worth 50 points each, strawberrys are worth 75 points each, kiwis are worth 100 points each and cherries are worth 150 points each.

Similarly, we can create a list of the images as follows:

```
pics = []
pics.append(pygame.image.load("apple.jpg"))
pics.append(pygame.image.load("strawberry.jpg"))
pics.append(pygame.image.load("kiwi.jpg"))
pics.append(pygame.image.load("cherry.jpg"))
```

Notice the same correlation between indices as previously described.

One might be asking why we store these values separate to the items themselves. The reason is that these values are shared amongst all fruits of the same type. If we have 100 apples and for each one store its point value in a list and its image (which is large), then we are wasting quite a bit of memory. Instead, the only important difference we have to store about each fruit "object" is which fruit it is, 0, 1, 2 or 3. Once we know that, we can use that information to look up how much it's worth or draw the appropriate image for it on the display surface.

If we wanted a unique image and point value for each fruit, even of the same kind, then it would make sense to add these items to the list that stores a single fruit.

IV. Checking to see if we clicked on a fruit

To see if we've clicked on a fruit, assume we have the following information:

1. The x and y coordinates of the mouse click.

2. The x and y coordinates of the top left corner of the fruit picture.

3. The width and height of the fruit picture.

For our function, the location of the mouse click is stored in a list called mypos, with mypos[0] being the x coordinate of the mouse click and mypos[1] being the y coordinate of the mouse click. All of the information about a single fruit for our function is stored in a list called f. f[0], f[1] are the coordinates of the top left corner of the fruit. f[4] stores which fruit it is (0, 1, 2 or 3). The list pics stores information about each of the corresponding pictures. In particular, we can obtain the width and height of a picture with the function get_width() and get_height() respectively.

Our basic strategy with our function is to screen out cases where we definitively haven't hit the fruit. Once we screen all of these out of the way, the only conclusion left is that we hit it. Here are the situations we haven't hit the fruit:

1. Our mouse click is above (lower y value) than the top of the fruit.

2. Our mouse click is to the left (lower x value) than the left side of the fruit.

3. Our mouse click is to the right (higher x value) than the right side of the fruit.

4. Our mouse click is below (higher y value) than the bottom of the fruit.

In the implementation chosen, we check for #1 and #2 in the same if statement, returning false if either is true. Then we check for #3 in the second clause of the if statement. Finally, we check for the opposite of #4 and return its value, since if we get this far in the code, this one check finalizes the answer.

Here is the implementation of the hit function:

```
def hit(f, mypos, pics):

    if mypos[0] < f[0] or mypos[1] < f[1]:
        return False

    if mypos[0] >= f[0] + pics[f[4]].get_width():
        return False

    return mypos[1] < f[1] + pics[f[4]].get_height()
```

The first if statement is the most straight-forward, since f stores the top left corner information of the fruit in question.

In the second if, we don't have direct access to the x pixel value of the right of the picture. But, we can obtain it from the x pixel value of the left of the picture (f[0]) as well as the width of the picture. Recall that f[4] is 0, 1, 2 or 3, depending on if the fruit is an apple, strawberry, kiwi or cherry. Then, this number is used as an index to the pics array, which will obtain the appropriate picture. Finally, for any picture, the get_width() function obtains the width of that picture.

The last statement directly does a return. If the y coordinate of the mouse click is less than the bottom of the picture, we know that we must have clicked on the picture, because to get to this portion of the function, the two previous if statements must be false, and if they are false, the click occurred below the top of the picture, to the right of the left side of the picture and not to the right of the right side of the picture. Alternatively, if this Boolean expression is false, then we know that the click occurred below the bottom of the picture so the picture was not captured.

V. Generating New Fruit Randomly

Now, we'll generate fruit randomly similar to how we generated raindrops in the second raindrop example. We'll make one new fruit every 10 frames. Each fruit will move from the top of the screen to the bottom, but it might also have a dx component, shifting it left or right. Finally, we will randomly select which fruit each new one is (0, 1, 2 or 3). Here is the code:

```
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)
    fruit.append([x,0,mydx,mydy,which])
```

This code can be adjusted in many ways based on what the programmer intends. In this particular example, we choose the place from the top the fruit drops completely randomly, as well as the fruit itself. We choose a random integer in between -2 to 2 to be dx, so the fruit goes at most 2 units to the left or right per frame. We choose a random integer in between 3 and 8 which means the fruit falls anywhere from 3 to 8 pixels per frame. Finally, we simply append a new list for this newly created fruit to our list of fruits.

VI. Counting Dropped Items and Ending the Game

In our rain example, we would remove useless items. For this game however, we want to count how many useless items were removed each time. This count adds to the number of "dropped fruit." We adjust our removeUseless function as follows to calculate and return this count:

```
def removeUseless(items):
    total = 0
    for item in items:
        if item[1] > SCREEN_H:
            items.remove(item)
            total += 1
    return total
```

All we had to add to our old function was a variable total, which we increment for any item that we remove in the if statement. Finally, we just return the value of total.

Now, we must correspondingly change how we call removeUseless, to do something with this value that it returns. In addition to that, we must detect the end of the game (when the number of dropped fruit exceeds 20) and quit at that time:

```
dropped += removeUseless(fruit)

if dropped > 20:
    print("Sorry, you have dropped more than 20 fruits.")
    print("The game is over.")
    print("Your score is",score)
    pygame.quit()
    sys.exit()
```

The whole program, with comments is included on the following pages.

```
# Arup Guha
# 7/15/2015
# Example to illustrate a mouse event.
# This is the fruit game - click on fruit to get points!!!

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

# Useful Constants
SCREEN_W = 1000
SCREEN_H = 600

# This function handles moving each item listed in items.
def move(items):
    for item in items:
        item[0] += item[2]
        item[1] += item[3]

# This function removes all items that will never be visible again,
# and returns how many were removed.
def removeUseless(items):
    total = 0
    for item in items:
        if item[1] > SCREEN_H:
            items.remove(item)
            total += 1
    return total

# Basic Set Up
pygame.init()
DISPLAYSURF = pygame.display.set_mode((SCREEN_W, SCREEN_H))
pygame.display.set_caption("Catch the fruit!")
WHITE = pygame.Color(255,255,255)
BLUE = pygame.Color(0,0,255)
clock = pygame.time.Clock()

# Visible fruit will be stored here.
fruit = []

# Store images here.
pics = []
pics.append(pygame.image.load("apple.jpg"))
pics.append(pygame.image.load("strawberry.jpg"))
pics.append(pygame.image.load("kiwi.jpg"))
pics.append(pygame.image.load("cherry.jpg"))

# How much each fruit is worth!
pts = [50, 75, 100, 150]
```

```
# Returns true iff mypos is within the picture specified by f.
def hit(f, mypos, pics):

    if mypos[0] < f[0] or mypos[1] < f[1]:
        return False
    if mypos[0] >= f[0] + pics[f[4]].get_width():
        return False
    return mypos[1] < f[1] + pics[f[4]].get_height()

curT = time.clock()

score = 0
dropped = 0
step = 0

# Main game loop starts here.
while True:

    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 hit(f, event.pos, pics):
                        score += pts[f[4]]
                        fruit.remove(f)

    # These images are big, so I only add a drop once every 10 time steps.
    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)
        fruit.append([x,0,mydx,mydy,which])

    DISPLAYSURF.fill(WHITE)

    # blit allows us to draw a surface onto another surface.
    for item in fruit:
        DISPLAYSURF.blit(pics[item[4]],(item[0], item[1]))

    pygame.display.update()

    # Move the drops for the next iteration and remove useless ones.
    move(fruit)
    dropped += removeUseless(fruit)
```

```
# Game ends!
if dropped > 20:
    print("Sorry, you have dropped more than 20 fruits.")
    print("The game is over.")
    print("Your score is",score)
    pygame.quit()
    sys.exit()

clock.tick(30)
step += 1
```