UNIVERSITY OF Hull

Writing Games with Pygame

Wrestling with Python

Getting Started with Pygame

- What Pygame does
- Getting started with Pygame
- Manipulating objects on the screen
- Making a sprite

UNIVERSITY OF Hull

Pygame overview

Pygame

- Pygame is a library of Python classes that you can use to create games
- We have already used Python libraries (for things like input from the user)
- Pygame is just another library that we can use to make games

Fetching Pygame

- Pygame might not be installed on your computer
- You can find it here:
  - http://pygame.org/download.shtml
- Follow the instructions for your platform and you should be able to use the library on your computer

Getting started with Pygame

Importing Pygame

```
import pygame
```

- Pygame is like any other Python library
- Before we can use it, we must import it into our program
- We do this at the very start

Initialising pygame

```
pygame.init()
```

- The game framework needs to do some setting up of stuff before you can use it to display the game
- You do this by calling the init method on the framework

What is going on here....

```
pygame.init()
```

- pygame is a class that we have imported from the pygame module

What is going on here....

```
pygame.init()
```

- pygame is a class that we have imported from the pygame module
- The pygame class contains a method called init which we need to call to start the pygame framework running
- If we forget to do this the game won't work

The pygame class

```
pygame.other stuff goes here
```

- There are other things that we use that are supplied as part of the pygame class
- It is essentially the container of all the game related things that we want to use

Getting a drawing surface

```
surface = pygame.display.set_mode((320,240))
```

- The first thing we need to do is get something to draw on
- We could call it the drawing surface
  - Although the name isn't significant
- This will be drawn on when the game runs

Getting a drawing surface

```
surface = pygame.display.set_mode((320,240))
```

- The pygame object also contains another object
  - We have seen that we can put one object inside another
- The pygame object actually contains a bunch objects that look after the game

Getting a drawing surface

```
surface = pygame.display.set_mode((320,240))
```

- The `set_mode` method is part of the display object
- It sets the display mode we want to use
  - And it returns a drawing surface that our game can use

Getting a drawing surface

```
surface = pygame.display.set_mode((320,240))
```

- Any ideas what these numbers mean?

Getting a drawing surface

```
surface = pygame.display.set_mode((320,240))
```

- Any ideas what these numbers mean?
- Yep, they are the width and the height of the screen that we want
- In the call above we want a screen that is 320 pixels wide and 240 pixels high

Pixel digression

- A pixel is the size of a dot on the display
- The more dots you have, the better quality the display
- You also find pixel dimensions when you are talking about camera and video screen resolution
- 320x240 is not particularly high resolution, but it will do for now

Why are there two brackets?

```
surface = pygame.display.set_mode((320,240))
```

- You may be wondering why there are two brackets at each end of the parameters
- Why doesn't `set_mode` just accept a pair of numbers which are the width and the height?
- This is because the dimensions of the screen are given as a *tuple*

What is a tuple?

```
size = (800,600)
```

- A *tuple* is a way of lumping together a number of values to make a sequence of values
- In this case we want to lump together the width and the height to give us a screen size

Tuples vs. Lists

- You might be thinking that you have seen something like a tuple before
- We have previously used *lists* to store collections of things
- So you might be wondering why we have the two things, and what the difference is

Tuples vs. Lists

- List
  - Everything in a List is the same kind of thing
  - We might have a list of customers or of ages, heights, or sales
- Tuple
  - A tuple brings together a collection of different things
  - A size tuple could hold the width and the height of the surface

Tuples and Python code

```
size = (800,600)
surface = pygame.display.set_mode(size)
```

- We can use a tuple anywhere we can use any other kind of python value (int or string for example)
- The code above makes a tuple called `size` and then creates a screen of that size

Setting the draw surface properties

```
pygame.display.set_caption("Awesome game")
```

- You can set the title of your game window to any string of text that you like by using the `set_caption` method
- Note that `set_caption` is another method exposed by the `display` object in `pygame`

Getting a drawing surface

PRACTICAL BREAK 1

Drawing a line on the screen

- Now that we have our display surface we can draw on it
- We are going to start by drawing a line
- Later on we will draw more interesting things

What defines a line?

- For Pygame to draw a line it needs to know a few things:
  - The surface to draw the line on
  - The colour of the line (red, green and blue)
  - The start of the line (xstart,ystart)
  - The end of the line (xend,yend)

Drawing the line

```
pygame.draw.line(surface,(255,0,0),(0,0),(50,20))
```

- The pygame object contains a draw object which is in charge of drawing stuff
- It can draw lots of things, including lines
- Lines are drawn by the line method
- This would draw a red line from 0,0 to 50,20

How many tuples can you see?

```
pygame.draw.line(surface,(255,0,0),(0,0),(50,20))
```

- The first parameter is the surface, which makes sense, but the rest are all tuples
- There are three tuples in the call above
- We can look at each one in turn

How many tuples can you see?

```
pygame.draw.line(surface,(255,0,0),(0,0),(50,20))
```

- This is the tuple that defines the colour
- It gives the colour of the line as three integers expressing red, green and blue levels
- Pygame uses 8 bits to express the intensity of each of the primary colours in the line
- 255 means the brightest possible

How many tuples can you see?

```
pygame.draw.line(surface,(255,0,0),(0,0),(50,20))
```

- This is the tuple that defines the start
- It gives the x and the y values for the coordinate that defines the start of the line
- These are expressed in pixels
- The origin is the top left hand corner

How many tuples can you see?

```
pygame.draw.line(surface,(255,0,0),(0,0),(50,20))
```

- This is the tuple that defines the end
- It gives the x and the y values for the coordinate that defines the start of the line

Flipping buffers

```
pygame.display.flip()
```

- If you just called the function above you would not see a red line – sad face ☹
- This is because pygame does the drawing on a back buffer
- You have to call the flip method to copy the back buffer to the foreground

Back buffers and flipping

- A game will repeatedly redraw the image on the screen as the game is being played
  - It will do this many time a second
  - The user should not see the drawing process
- The display system only wants to update the user screen when all the drawing for a frame has been finished
  - The flip method is used to trigger this update

Drawing some lines

PRACTICAL BREAK 2

Getting a Game Going

- We now know how we can draw things in games
- But we would not do this from the shell prompt
- Instead we'd want to create something that contained the game program

A class to run a pygame program

```
import pygame
import random

class DrawingFun:
    def drawLines(self):
        pygame.init()
#       rest of line drawing here

fun = DrawingFun()
fun.drawLines()
```

- This is how we get a game program going
- Create an instance of a class and then call a method to run the game

A class to run a pygame program

```
import pygame
import random

class DrawingFun:
    def drawLines(self):
        pygame.init()
#       rest of line drawing here

fun = DrawingFun()
fun.drawLines()
```

- These are the imports we are using
- I want to use pygame and some random numbers

A class to run a pygame program

```
import pygame
import random

class DrawingFun:
    def drawLines(self):
        pygame.init()
#       rest of line drawing here

fun = DrawingFun()
fun.drawLines()
```

- This creates a class called `DrawingFun`
- It will contain our "game" code

A class to run a pygame program

```
import pygame
import random

class DrawingFun:
    def drawLines(self):
        pygame.init()
#       rest of line drawing here

fun = DrawingFun()
fun.drawLines()
```

- This creates a method called `drawLines`
- It will draw some lines for us

A class to run a pygame program

```
import pygame
import random

class DrawingFun:
    def drawLines(self):
        pygame.init()
#       rest of line drawing here

fun = DrawingFun()
fun.drawLines()
```

- The variable `fun` is a reference to an instance of the `DrawingFun` class

A class to run a pygame program

```
import pygame
import random

class DrawingFun:
    def drawLines(self):
        pygame.init()
#       rest of line drawing here

fun = DrawingFun()
fun.drawLines()
```

- This is the call of `drawLines` that makes the magic happen

Class pattern

- You will see this pattern very frequently
- We are using a class as a container for our game and putting methods (and data) into the class to hold the game information
- Then we start the game running by calling a method to get things going.

Drawing loads of lines

PRACTICAL BREAK 3

Next Time

- Next time we will see how to construct objects that we can move around the screen
- We will also consider how we can get movement and animation, along with control inputs
- It's going to be fun.....