1. Introduction to PyGame

By Thuan Ta

Very briefly, Python is a very nice programming language for making things quickly with minimal overhead. There's no need to compile code because it's generally done on-the-fly and unlike C/C++, there's no need to worry about memory management. PyGame is a Python multimedia library written on top of SDL that can be used to make games quickly!

1.1 Setting up

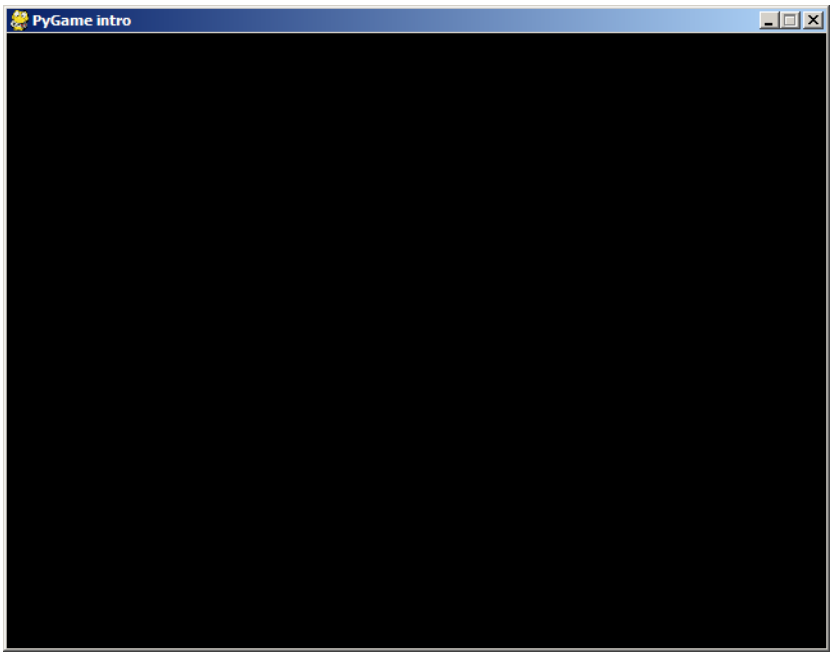
1) Download and install the Python programming language software: http://www.python.org/download/. As of this writing, the latest Python version is 2.5.1.
2) Download and install PyGame from http://www.pygame.org/download.shtml. The latest version is 1.7.1 and make sure you download the PyGame binary for your operating system that has been compiled for Python 2.5.
3) For writing code, here are a couple of free Integrated Development Environments (IDEs) which I recommend:
    - WingIDE 101: http://wingware.com/downloads/wingide-101/installers
    - Eclipse: http://www.eclipse.org/downloads/ (Download the Eclipse classic)
        - You'll also need the PyDev Python plug-in for Eclipse: http://pydev.sourceforge.net/download.html

1.2 Creating your first game in Python!

Load up your favorite IDE and type in the following code (Python is sensitive to indention, so be sure to include them!):

```
import pygame

from pygame.locals import *

screen_mode = (640, 480)
color_black = 0,0,0


class Game:

    # this gets called first
    def __init__(self):

        pygame.init()
        self.screen = pygame.display.set_mode(screen_mode)
        pygame.display.set_caption("PyGame intro")

        self.quit = False

    # put game update code here
    def update(self):

        return

    # put drawing code here
    def draw(self):

        self.screen.fill(color_black)
        pygame.display.flip()

    # the main game loop
    def mainLoop(self):

        while not self.quit:

            # handle events
            for event in pygame.event.get():

                if event.type == QUIT:
                    self.quit = True


            self.update()
            self.draw()


if __name__ == '__main__' :

    game = Game()
    game.mainLoop()
```

Once you have typed (or copy and pasted) the code correctly and run it, you'll get this:



A 640x480 game window in black! Now an explanation of the code:

```
import pygame

from pygame.locals import *
```

These two lines simply say you want to use the PyGame library in your project. You'll always have to include these near the top of your file whenever you want to use PyGame.

```
screen_mode = (640, 480)
color_black = 0,0,0
```

`screen_mode` is a variable we have declared which we'll use for our game to set the screen mode to 640 x 480 while `color_black` is a 3-tuple variable representing black as an RGB value 0,0,0.

```
class Game:
```

We are defining a Game class here. From Wikipedia:

" In object-oriented programming, a class is a programming language construct used to group related instance variables and methods. "

Because of the large number of potential variables all of our methods will have to access in our game (e.g. the screen, game status, sprites, etc.), it's easier to define a class with member variables which all of our methods can access rather than making the variables global.

```
    # this gets called first
    def __init__(self):

        pygame.init()
        self.screen = pygame.display.set_mode(screen_mode)
        pygame.display.set_caption("PyGame intro")
```

```
        self.quit = False
```

Here is the constructor of our Game class, and it's the first method that will always get executed whenever we create an instance of the Game class. We generally want to do all our initialization here. First initialize the PyGame library by calling `pygame.init()`. After initializing PyGame, we can then ask it to create a window for us for our game with a call to `pygame.display.set_mode()` and passing in the screen mode parameter as a 2-tuple, which we defined earlier. `set_mode()` will return a display surface (a *surface* is basically a chunk of memory that can be thought of as a drawing canvas for images). The display surface is where we will be doing all of our drawing and displaying it on the screen, so it's important that we make a member variable for it. We can set our game to have a window caption by calling `pygame.display.set_caption()` with the name we want, in this case "PyGame intro". Finally we declare a member variable quit to let us know if it is time to quit the game or not.

```
    # put game update code here
    def update(self):

        return

    # put drawing code here
    def draw(self):

        # clear the screen
        self.screen.fill(color_black)

        # add code here

        # display updated scene
        pygame.display.flip()
```

The two methods `update()` and `draw()` is where you'll want to add code for your future game. `update()` should contain code that's related to updating the state of the game (e.g. moving your player and enemies positions, updating physics, etc.) while `draw()` is where you'll want to add code relating to drawing sprites and other images. At the beginning of the `draw()` function, you'll generally want to clear the screen by calling `self.screen.fill()` with your desired fill colour (in this case black) before you begin drawing your next scene. Otherwise you'll be drawing a "used canvas" which won't look right when you display it. At the end of your `draw()` function, you must make a call to `pygame.display.flip()` or else your updated scene won't be shown; you'll still have the old scene which will look like your game isn't moving at all.

```
    # the main game loop
    def mainLoop(self):

        while not self.quit:

            # handle events
            for event in pygame.event.get():

                if event.type == QUIT:
                    self.quit = True


            self.update()
            self.draw()
```

Once initialization is done, the main "game loop" executes which forever repeats until the user has decided to quit. The sequence is basically: check for events/input, update game state, draw, and repeat until user quits. The game loop is the key concept for all games regardless of what programming language and framework you use. The only event we check for here is if the user quits by closing the window. If the user hasn't, then we continue to update and draw.

```
if __name__ == '__main__' :

    game = Game()
    game.mainLoop()
```

And lastly, this is how we code our game to run. Now that you have an idea of how PyGame works, do me a favour and try changing the game background to CornflowerBlue. (Hint: the RGB value of CornflowerBlue is 100, 149, and 237). ☺

1.3 Loading and drawing images

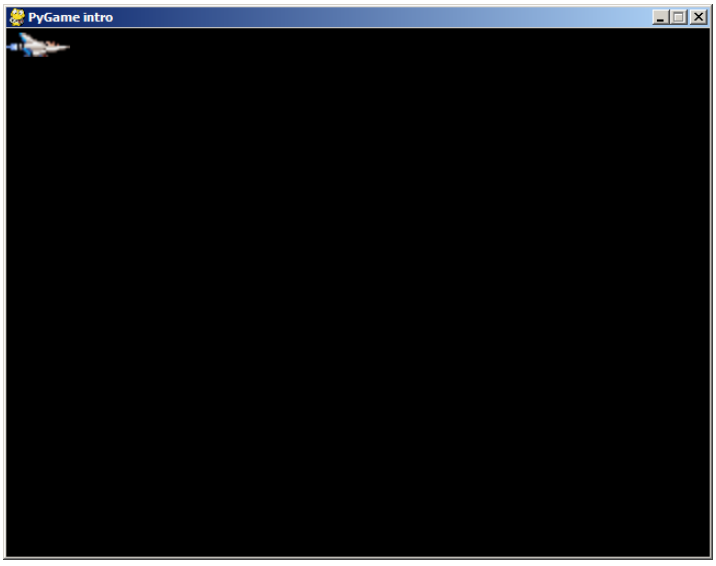
Now let's try to load an image file and draw it on the screen. I have an image file called exalter.bmp that I want to display. In the constructor method, add the following to the end of the method:

```
self.image = pygame.image.load("exalter.bmp")
```

And in the draw method add:

```
self.screen.blit(self.image, (0,0))
```

What we're doing is basically asking PyGame to load the image file to variable image. In draw(), we draw it to the screen by telling the screen surface to blit the image onto itself (aka the destination surface) at pixel 0,0. *Blitting* is a computer graphics term used to describe the operation of copying images from one onto another. Now run the game and you'll get something like:



Neato.

1.4 Getting mouse and keyboard input

Let's make our game more exciting by moving the Exalter with the mouse, because it's real easy to do! PyGame has a module which helps you get input for the mouse. First, add these member variables to your Game class in the constructor like so:

```
self.x = 320

self.y = 240
```

Then modify the blitting code in `draw`() to:

```
self.screen.blit(self.image, (self.x, self.y))
```

Add the next line of code in your update() method:

```
(self.x, self.y) = pygame.mouse.get_pos()
```

That's it. Now run the game and volia! The ship moves along with your mouse. `pygame.mouse.get_pos()` basically gets the current (x,y) position of the mouse relative to the top left corner of the window and returns the (x,y) coordinates as a tuple. If you only want to know by how many pixels the mouse has moved since the last call, use `pygame.mouse.get_rel()`instead.

PyGame also has a keyboard module which helps you get input from the keyboard and you can write code to respond to those keyboard events. There are a two main ways of getting keyboard input: through the keyboard module and via the event pump. We'll only look at using the keyboard module first and try to make the Exalter move with it. Again in your update `method`(), get rid of or comment out the mouse code and add the following:

```
keys = pygame.key.get_pressed()

if keys[K_LEFT]:
    self.x = self.x - 1
if keys[K_RIGHT]:
    self.x = self.x + 1
```

Can you guess what the code does? Assuming you're right, yes, it moves the image (aka sprite) left when you hold down the LEFT arrow and right when you hold down the RIGHT arrow keys. ☺ Now see if you can move it up and down.

1.5 Very basic timing

This will be the last topic for today. When you moved the ship with the arrow keys, did you notice how fast the ship moved? This is because the game is running as fast as it can on your computer. In other words, there isn't any timing. Obviously we need timing for games to be playable, and PyGame provides a very simple way of locking your game to a certain frame rate. PyGame, like most multimedia/game libraries, provides a timer module. We should first create an instance of a Clock object in our Game class. Add this in the constructor:

```
self.clock = pygame.time.Clock()
```

Now we can lock the frame rate, to say 60 fps, by adding:

```
        self.clock.tick(60)
```

Since this kind of work isn't quite like updating the game state or drawing, I typically add this near the end of the `mainLoop()` method, after having done `update()` and `draw()`. Now run the game and guess what has happened…

1.6 Further reading

I hope this was a brief but helpful introduction to using PyGame. As you can see, the number of lines of code you need to write to get something up and running is very little, and I don't know of any other language and game library that'll let you do this much with very few lines of code. By the end of this tutorial, the amount of code I've written is a little under 75 lines! There's still a lot more which you can do in PyGame, like playing sounds, CDROM or MP3s, using a joystick, and displaying text. If you're dying to learn more, here are some links to satisfy your curiosity:

- PyGame documentation & tutorials: http://www.pygame.org/docs/
- More tutorials on their wiki: http://www.pygame.org/wiki/tutorials
- Even more tutorials and links to free art, sounds, fonts, etc.: http://www.pygame.org/wiki/resources

Appendix

Here's the final code that you should have at the end of this tutorial.

```
import pygame

from pygame.locals import *

screen_mode = (640, 480)
color_black = 0,0,0

class Game:

    # this gets called first
    def __init__(self):

        pygame.init()
        self.screen = pygame.display.set_mode(screen_mode)
        pygame.display.set_caption("PyGame intro")
        self.quit = False

        self.image = pygame.image.load("exalter.bmp")
        self.x = 320
        self.y = 240

        self.clock = pygame.time.Clock()

    # put game update code here
    def update(self):

        keys = pygame.key.get_pressed()
```

```
        if keys[K_LEFT]:
            self.x = self.x - 1
        if keys[K_RIGHT]:
            self.x = self.x + 1

        #(self.x, self.y) = pygame.mouse.get_pos()

        return

    # put drawing code here
    def draw(self):

        # clear the screen
        self.screen.fill(color_black)

        # add code here
        self.screen.blit(self.image, (self.x, self.y))

        # display updated scene
        pygame.display.flip()

    # the main game loop
    def mainLoop(self):

        while not self.quit:

            # handle events
            for event in pygame.event.get():

                if event.type == QUIT:
                    self.quit = True


            self.update()
            self.draw()
            self.clock.tick(60)


if __name__ == '__main__' :

    game = Game()
    game.mainLoop()
```