Pygame!

Python User Group Freiburg

Felix Hoffmann

http://about.me/felixhoffmann

December 12, 2013

`http://www.pygame.org`

Pygame is a Python wrapper around the SDL library (Simple DirectMedia Layer) with a few unique libraries with an emphasis on game programming, written by Pete Shinners.

From the FAQ

Does not require OpenGL

Uses either opengl, directx, windib, X11, linux frame buffer, and many other different backends... including an ASCII art backend!

Truly portable

Supports Linux, Windows, Windows CE, BeOS, MacOS, Mac OS X, FreeBSD, NetBSD, OpenBSD, BSD/OS, Solaris, IRIX, and QNX . . .

Silliness built in!

A simple Pygame example

```python
import pygame

pygame.init()
screen = pygame.display.set_mode((640, 480))

color = [(0,0,0),(255,255,255)]
running = True

while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
        if event.type == pygame.KEYDOWN:
            color[0], color[1] = color[1],color[0]

    screen.fill(color[0])
    pygame.display.flip()
```

`01-simple_pygame.py`

What each element does: Importing & Initializing

```
import pygame
```

to import the Pygame module.

```
from pygame.locals import *
```

Optional. Puts limited set of constant and function in the **global namespace**.

```
pygame.init()
```

to initialize Pygame's modules (e.g. `pygame.font`). Not always needed, but recommended in *any* case.

What each element does: Setting Window & Screen

```
screen = pygame.display.set_mode((640, 480))
```

initializes a **window** with dimensions 640×480 and returns the **screen object**.

Everything to be displayed needs to be drawn on the `screen`.

Initializing a Pygame window

Together:

```python
import pygame

pygame.init()
screen = pygame.display.set_mode((640, 480))
```

`02-window.py`

Initializing a Pygame window - Extended

```python
import pygame

pygame.init()
screen = pygame.display.set_mode((640, 480))

running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
```

What each element does: The Main Loop

```
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
        if event.type == pygame.KEYDOWN:
            color[0], color[1] = color[1],color[0]

    screen.fill(color[0])
    pygame.display.flip()
```

is the **main loop** of the game.

- listen to events $\rightarrow$ respond
- proceed the game
- draw on the screen
- stop when done

A framework for your Pygames

```python
import pygame

pygame.init()
screen = pygame.display.set_mode((640, 480))

running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
        if event.type == pygame.KEYDOWN:
            react_to_user_input()

    do_things_the_game_does()
    draw_everything_on_the_screen()
```

Next:

- Drawing
- User Input
- Game Events

Drawing

Drawing on the screen I

Filling the screen with a color:

```python
blue = (0,0,255)
screen.fill(blue)
pygame.display.flip()
```

After all drawing is done, call `display.flip()` to **update** the display.

Use pygame.draw to draw geometric shapes. A circle:

```python
red = (255,0,0)
# position (320,240), radius = 50
pygame.draw.circle(screen, red, (320,240), 50)
```

Drawing on the screen II

Geometric shapes available for `pygame.draw`:

```
circle(Surface, color, pos, radius, width=0)
polygon(Surface, color, pointlist, width=0)
line(Surface, color, start, end, width= 1)
rect(Surface, color, Rect, width=0)
ellipse(Surface, color, Rect, width=0)
```

Example:

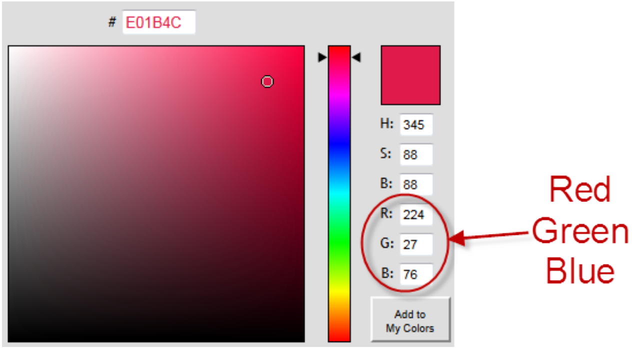
```
red = (255,0,0)
pygame.draw.line(screen, red, (10,50),(30,50),10)
```

Drawing on the screen - Colors

Defining a color

```
gray = (200,200,200)
#(red, green, blue)
```

Use for example `colorpicker.com`:



E01B4C

H: 345
S: 88
B: 88
R: 224
G: 27
B: 76

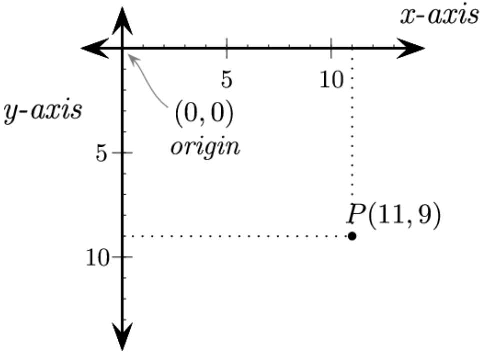
Source: Paul Craven, programarcadegames.com/

Drawing on the screen - Positions

Defining a position:

```
P = (11,9)
#(x-axis, y-axis)
```

To the reference coordinate system



Source: Paul Craven, programarcadegames.com/

Drawing on the screen - Rects

```
pygame.Rect(left, top, width, height)
```

to create a Rect.

```
box = pygame.Rect(10, 10, 100, 40)
pygame.draw.rect(screen, blue, box)
#draws at (10,10) rectangle of width 100, height 40
```

Rect anchors:

```
top, left, bottom, right
topleft, bottomleft, topright, bottomright
midtop, midleft, midbottom, midright
center, centerx, centery
size, width, height
w,h
```

A full drawing example

```python
import pygame

pygame.init()
screen = pygame.display.set_mode((640, 480))

white = (255,255,255)
blue = (0,0,255)

running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    screen.fill(white)
    pygame.draw.circle(screen, blue, (320,240), 100)
    # position (320,240), radius = 100

    pygame.display.flip()
```

`04-drawing.py`.

User Input

Events

get all events in Pygame's event queue

```
pygame.event.get()
```

usually used as

```
for event in pygame.event.get():
    if event.type == YourEvent:
        react_to_your_event()
```

Event types

Some of the most important event types are:

```
pygame.QUIT
pygame.KEYDOWN
pygame.KEYUP
pygame.USEREVENT
```

With

```
    from pygame.locals import *
```

from earlier, prefix `pygame` isn't needed.

Events

React to KEYDOWN event:

```
while running:
    for event in pygame.event.get():
        if event.type == KEYDOWN:
            react_to_key()
```

Which key? $\rightarrow$ if event type is KEYDOWN or KEYUP event has attribute **key**.

```
for event in pygame.event.get():
    if event.type == KEYDOWN:
        if event.key == K_ESCAPE:
            running = False
```

Pygame Keys

Some of the most important keys are:

```
K_RETURN
K_SPACE
K_ESCAPE
K_UP, K_DOWN, K_LEFT, K_RIGHT
K_a, K_b, ...
K_0, K_1, ...
```

Full list of keys: `http://www.pygame.org/docs/ref/key.html`

Getting continuous input

`KEYDOWN` is a unique event.

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

to get keys currently pressed.

```
if key[pygame.K_UP]:
    move_up()
```

to check and react on a specific key.

A user input example

```python
color = [0,0,0]

while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
        if event.type == KEYDOWN and event.key == K_SPACE:
            color = [0,0,0]

    keys = pygame.key.get_pressed()
    if keys[K_UP]:
        color = [(rgb+1)%256 for rgb in color]

    screen.fill(color)
    pygame.display.flip()
```

`05-user_input.py`

Pygame’s Clock

Limiting the frames per second

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

to initialize the clock.

In your main loop call

```
clock.tick(60) #limit to 60 fps
```

Clock example

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

while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
        if event.type == KEYDOWN and event.key == K_SPACE:
            color = [0,0,0]

    keys = pygame.key.get_pressed()
    if keys[K_UP]:
        color = [(rgb+1)%256 for rgb in color]

    screen.fill(color)
    pygame.display.flip()

    clock.tick(60)
```

`06-clock.py`

Game Events

Sprites

Pygame's **sprite class** gives convenient options for handling interactive graphical objects.

If you want to draw **and** move (or manipulate) and object, make it a sprite.

Sprites

can be grouped together

are easily drawn to a surface (even as a group!)

have an update method that can be modified

have collision detection

Basic Sprite

```python
class SpriteExample(pygame.sprite.Sprite):

    def __init__(self):
        pygame.sprite.Sprite.__init__(self)

        self.image = #image
        self.rect = #rect

    def update(self):
        pass
```

Sprite from a local image

```python
class SpriteExample(pygame.sprite.Sprite):

    def __init__(self):
        pygame.sprite.Sprite.__init__(self)

        self.image = pygame.image.load('local_img.png')
        self.rect = self.image.get_rect()
        self.rect.topleft = (80,120)

    def update(self):
        pass
```

Self-drawn sprites: Rectangle

```python
class Rectangle(pygame.sprite.Sprite):

    def __init__(self):
        pygame.sprite.Sprite.__init__(self)
        self.image = pygame.Surface([200, 50])
        self.image.fill(blue)
        self.rect = self.image.get_rect()
        self.rect.top, self.rect.left = 100, 100

    def update(self):
        pass
```

Sprites: A reminder about classes

```python
class Rectangle(pygame.sprite.Sprite):

    def __init__(self, color):
        pygame.sprite.Sprite.__init__(self)
        self.image = pygame.Surface([200, 50])
        self.image.fill(color)
        self.rect = self.image.get_rect()
        self.rect.top, self.rect.left = 100, 100

    def update(self):
        pass
```

```python
white, blue = (255,255,255), (0,0,255)

white_rect = Rectangle(white)
blue_rect = Rectangle(blue)
```

Sprite groups

```
new_sprite_group = pygame.sprite.Group(sprite1)
```

to create a new sprite group containing sprite `sprite1`.

```
new_sprite_group.add(sprite2)
```

to add another sprite later.

Group updating and drawing:

```
#inside main loop:

new_sprite_group.update() #game events
new_sprite_group.draw()   #drawing
```

Framework for working with sprite groups

```python
class NewSprite(pygame.sp...
    #defining a new sprite

newsprite = NewSprite()
sprites = pygame.sprite.Group()
sprites.add(newsprite)

while running:
    for event in ...

    sprites.update() #game events
    sprites.draw()
    pygame.display.flip()
```

Sprite groups: working example

```python
class Circle(pygame.sprite.Sprite):
    def __init__(self):
        pygame.sprite.Sprite.__init__(self)
        self.image = pygame.Surface([100, 100])
        pygame.draw.circle(self.image, blue, (50, 50), 50)
        self.rect = self.image.get_rect()
        self.rect.center = [320,240]

def draw(sprites):
    screen.fill(white)
    sprites.draw(screen)
    pygame.display.flip()

circle = Circle()
sprites = pygame.sprite.Group(circle)

while running:
    sprites.update()
    draw(sprites)
```

07-sprite-circle.py

Working example: Transparency

By default `pygame.Surface` is black. For transparency:

```python
class Circle(pygame.sprite.Sprite):
    def __init__(self):
        pygame.sprite.Sprite.__init__(self)
        self.image = pygame.Surface([100, 100], pygame.SRCALPHA, 32)
        pygame.draw.circle(self.image, blue, (50, 50), 50)
        self.image = self.image.convert_alpha()
```

`08-circle.py`

Sprite Collision Detection

```
pygame.sprite.collide_rect(left, right)
```

to detect collision between two sprites

```
pygame.sprite.spritecollideany(sprite, group)
```

to test if the given sprite intersects with any sprites in a Group

Final slide

Pygame's documentation:

`http://www.pygame.org/docs/`

Richard Jones - Introduction to Pygame:

- video:

  `http://www.youtube.com/watch?v=mTmJfWdZzbo`
- code samples:

  `https://bitbucket.org/r1chardj0n3s/pygame-tutorial/src`

Happy Holidays! :)