

Posts Shouts Comunidades Juegos Tops ▾



INGRESAR

REGISTRARTE

⌚ 3 años Taringa! Linux Como programar un juego...



Me gusta 478 019 Seguir

Como programar un juego paso a paso con Pygame



A favoritos



Ir a comentarios

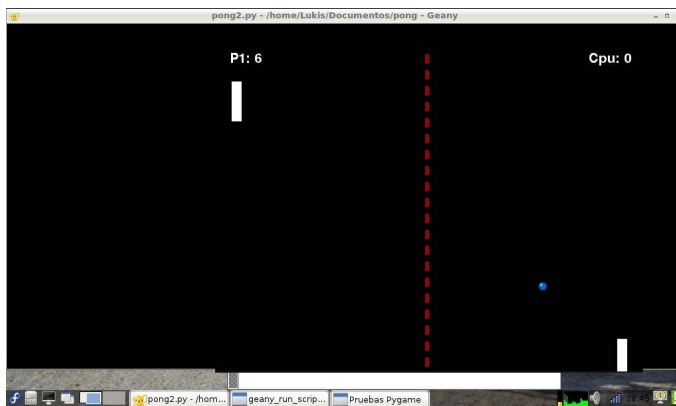


Compartir vía mail

f Compartir

Twitter

Ayer aburrido en casa me dieron ganas de ponerme a ver como programar juegos (algo que siempre tuve pendiente) y encuentre un tutorial bastante interesante que explica paso a paso como crear un juego usando como ejemplo el viejo clasico "Pong".



El tutorial es tan largo como meticuloso, pero tambien facil de seguir y practico. En 2 horas ya tenia el juego funcionando (bueno, convengamos que tampoco es ninguna maravilla... 😊).

Antes de pasar al tutorial veamos que necesitamos:

- 1) tener Python instalado
- 2) tener Pygame instalado
- 3) algun editor de texto o alguna IDE de desarrollo para escribir el codigo
- 4) imagenes sonidos y fuentes del juego (las dejo mas abajo cuando comienze el ejemplo)

Instalando Python y Pygame

Desde Windows:

Descargamos e instalamos en este orden:

<http://www.python.org/ftp/python/2.5.1/python-2.5.1.msi> (python)

<http://www.pygame.org/ftp/pygame-1.7.1release.win32-py2.5.exe> (pygame)

Desde GNU/Linux

En ubuntu

sudo apt-get install python-pygame

En Fedora



Maclu

28 Seguidores

1.712 Puntos

9 Posts

Avanzado

Posts Relacionados



Cómo construir la hamburguesa perfecta



El proyecto Debian Live muere de manera abrupta



¿ Crees que puedes aprender programación ?



15 Chistes Simples Pero Graciosos

```
sudo yum install python
sudo yum install pygame
```

En cuanto al IDE, en linux recomiendo Geany.

Pr

Directorio base e imagenes

Lo primero que necesitamos es un lugar de trabajo donde tengamos todos nuestros archivos y recursos necesarios organizados. Yo tengo un directorio llamado pygame donde voy metiendo una carpeta para cada proyecto que hago.

Os dejo la carpeta inicial del juego que vamos a realizar en este tutorial. Contiene un fichero base(explicado mas abajo) y una carpeta images que contiene las imagenes y una fuente tipografica que usaremos para realizar nuestro juego.

[pong.zip](#)

El documento base

Mi metodo de trabajo es tener todo organizado para que el codigo sea facil de leer, es por eso que antes de empezar voy a compartir mi plantilla base que podeis guardar como plantilla.py. A partir de ella escribo mis programas y es muy util para tener siempre a mano.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

# Modulos

# Constantes

# Clases
# -----

# -----

# Funciones
# -----

# -----

def main():
    return 0

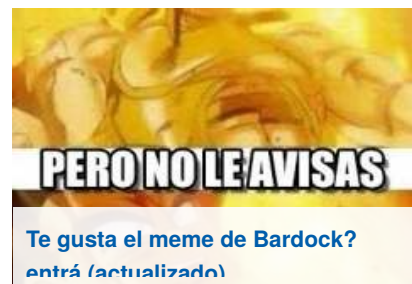
if __name__ == '__main__':
    main()
```

Indice:

- 1) Importar e inicializar.
- 2) Creando una ventana.
- 3) Cargar imagenes.
- 4) Creando Sprites.
- 5) Moviendo Sprites.
- 6) Control del teclado.
- 7) Colisiones.
- 8) Inteligencia Artificial.



Quieres programar en Android
pero no entiendes nada? No más!



9) Sistema de puntuacion.

1. Importar e inicializar

Lo primero que hay que hacer para trabajar con Pygame es importarlo e inicializarlo. Para importarlo basta con las lineas:

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

La primera linea importa pygame y la segunda carga las constantes para poder utilizar, por ejemplo, K_ESCAPE, en lugar de estar de tener que llamarla a traves del modulo Pygame. Una vez importado pasamos a inicializar. Se puede inicializar cualquiera de los modulos de Python por separado, pero para no complicarnos por ahora inicializaremos todo pygame con la linea:

```
pygame.init()
```

Esta se debe ejecutar antes de empezar a usar Pygame, un buen lugar es antes de llamar a la funcion main(), justo aqui:

```
if __name__ == '__main__':
    pygame.init()
    main()
```



2. Creando una ventana

Lo primero que necesitamos es definir las constantes del ancho y alto de la ventana, yo estas las defino como constantes porque de momento no vas a hacer cosas complicadas con cambios de resolucion ni nada por el estilo y tenerlas como constantes ayuda a trabajar con ellas desde cualquier clase o funcion.

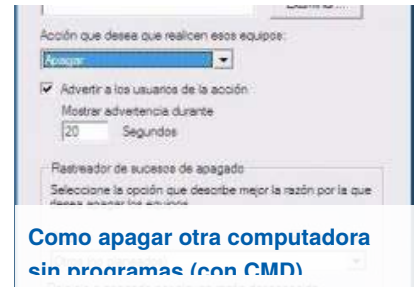
```
WIDTH = 640
HEIGHT = 480
```

Ahora pasamos a crear la ventana, se debe crear dentro de la funcion main(), es tan facil como lo siguiente:

```
screen = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("Pruebas Pygame")
```

La primera linea crea una ventana, date cuenta que la pasamos la tupla (WIDTH, HEIGHT) para definir las dimensiones de la ventana. La segunda linea sirve para definir el titulo de la ventana. El programa nos queda asi:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
```



```
# Modulos
import pygame
from pygame.locals import *

# Constantes
WIDTH = 640
HEIGHT = 480

# Clases
# -----

# -----

# Funciones
# -----

# -----

def main():
    screen = pygame.display.set_mode((WIDTH, HEIGHT))
    pygame.display.set_caption("Pruebas Pygame")
    return 0

if __name__ == '__main__':
    pygame.init()
    main()
```

Si lo ejecutas te daras cuenta que se crea una ventana, pero esta desaparece al instante. Esto es debido a que necesitamos crear un bucle infinito, que sera el bucle del juego. Lo creamos justo despues de la creacion de la ventana.

```
while True:
    pass
```

Como vemos es un bucle infinito vacio que se ejecuta eternamente y hace que la ventana no se cierra, pero ¡sorpresa!, ahora no hay forma de cerrar la ventana si no es matando el proceso. La solucion consiste en que el bucle infinito compruebe si queremos cerrar la ventana y en ese caso que deje de ejecutarse. Sustituimos las dos lineas anteriores por:

```
while True:
    for eventos in pygame.event.get():
        if eventos.type == QUIT:
            sys.exit(0)
```

La primera linea del bucle recorre con un `for pygame.event.get()` que es una lista interna de Pygame con los eventos que se estan ejecutando. En la siguiente linea comprobamos que si el tipo de evento es igual a `QUIT`, es decir, que si le estamos dando a la crucecita de cerrar ventana. En caso de que si, invoca a `sys.exit(0)`, para el que necesitamos haber importado el modulo `sys`, y cierra el programa. Ya tenemos la manera de crear una ventana y controlar cuando cerrarla.

El codigo nos queda asi:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

# Modulos
import sys, pygame
from pygame.locals import *

# Constantes
WIDTH = 640
```



**Como Reparar o Arreglar
Audifonos/Auriculares**



**Scioli se reunió con científicos y
Macri jugando al Basquet**



**5 verdades sobre la UFC que
nunca te dieron**



Si gana Macri saca 6,7,8



**Los dibujos animados de nuestra
infancia crecieron**


```

HEIGHT = 480

# Clases
# -----

# -----

# Funciones
# -----

# -----

def main():
    screen = pygame.display.set_mode((WIDTH, HEIGHT))
    pygame.display.set_caption("Pruebas Pygame")
    while True:
        for eventos in pygame.event.get():
            if eventos.type == QUIT:
                sys.exit(0)
        return 0

if __name__ == '__main__':
    pygame.init()
    main()

```

Gf

3. Cargar imagenes

Para cargar una imagen en Pygame vamos a usar una funcion que esta muy bien y facilita mucho el cargar imagenes al vuelo, pongo la funcion y la explicamos linea por linea:

```

def load_image(filename, transparent=False):
    try: image = pygame.image.load(filename)
    except pygame.error, message:
        raise SystemExit, message
    image = image.convert()
    if transparent:
        color = image.get_at((0,0))
        image.set_colorkey(color, RLEACCEL)
    return image

```

La **línea 1** define la funcion, recibe dos parametros el nombre/ruta del archivo y la segunda si tiene parte transparente (por defecto definida como falso).

La **línea 2** asigna a la variable imagen la imagen a traves de la funcion de Pygame `pygame.image.load()` si se puede sino, en la **líneas 3 y 4** manejan el error y salen del programa.

La **línea 5** convierte la imagen al tipo interno de Pygame que hace que sea mucho mas eficiente.

Las **línea 6** es un condicional que controla si el parametro `transparent` es verdadero y en caso afirmativo ejecuta las **líneas 7 y 8**, la primera obtiene el color del pixel (0, 0) de la imagen (esquina superior izquierda) y la la línea 8 lo define como color transparente de la imagen. Es decir que si quieres una imagen con transparencia, el color que actua como transparente se toma del pixel superior izquierdo, así que asegurate que este color no esta en la imagen.

Por ultimo la **línea 9** retorna la imagen despues de todo el proceso.



que componentes escoger para
armar tu propia pc



Los memes del debate
presidencial



Convierte tu celular en
reproductor de hologramas



Comen gatos: ¿la culpa es del
modelo?



¡El Acoso en Internet se esta
cagando la vida de esta chical!

El programa nos queda asi:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

# Modulos
import sys, pygame
from pygame.locals import *

# Constantes
WIDTH = 640
HEIGHT = 480

# Clases
# -----

# -----

# Funciones
# -----

def load_image(filename, transparent=False):
    try: image = pygame.image.load(filename)
    except pygame.error, message:
        raise SystemExit, message
    image = image.convert()
    if transparent:
        color = image.get_at((0,0))
        image.set_colorkey(color, RLEACCEL)
    return image

# -----

def main():
    screen = pygame.display.set_mode((WIDTH, HEIGHT))
    pygame.display.set_caption("Pruebas Pygame")
    while True:
        for eventos in pygame.event.get():
            if eventos.type == QUIT:
                sys.exit(0)
        return 0

if __name__ == '__main__':
    pygame.init()
    main()
```

Poniendo un fondo

Vamos a poner un fondo a nuestra ventana usando nuestra funcion load_image. Para cargar la imagen usaremos la siguiente linea, la ponemos justo antes de entrar en el bucle del juego:

```
background_image = load_image('images/fondo.png')
```

Como ves no definimos color transparente, porque para el fondo no es necesario.

Ahora necesitamos "ponerla en la ventana" para ello dentro del bucle principal debemos definir que imagen poner y en que posicion:

```
screen.blit(background_image, (0, 0))
```



chica de 17 años cumple su sueño al morir



Aprende a programar con Minecraft! - Inteligencia Colectiva



Los mejores memes del debate presidencial Scioli - Macri



Lana del Rey dejó que la violen sexualmente



inexplicable misterio de árboles doblados

Esto es lo que hace es poner la imagen la pantalla sobre las que habia, recibe dos parametros (en realidad algunos mas), la imagen a poner y en que lugar situar la esquina superior izquierda de la imagen, en este caso (0, 0), es decir, queremos que la esquina superior izquierda de la imagen este en la posicion (0, 0), con esto conseguimos que el fondo ocupe toda la ventana (si la imagen es mayor o superior a la ventana, claro esta).

por ultimo, como ultima linea del bucle debemos incluir la siguiente linea:

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

Esto lo que hace es actualizar toda la ventana para que se muestren los cambios que han sucedido.

El programa nos queda de la siguiente manera:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

# Modulos
import sys, pygame
from pygame.locals import *

# Constantes
WIDTH = 640
HEIGHT = 480

# Clases
# -----

# -----

# Funciones
# -----

def load_image(filename, transparent=False):
    try: image = pygame.image.load(filename)
    except pygame.error, message:
        raise SystemExit, message
    image = image.convert()
    if transparent:
        color = image.get_at((0,0))
        image.set_colorkey(color, RLEACCEL)
    return image

# -----

def main():
    screen = pygame.display.set_mode((WIDTH, HEIGHT))
    pygame.display.set_caption("Pruebas Pygame")

    background_image = load_image("images/fondo.png")

    while True:
        for eventos in pygame.event.get():
            if eventos.type == QUIT:
                sys.exit(0)

        screen.blit(background_image, (0, 0))
        pygame.display.flip()
        return 0

if __name__ == '__main__':
```



Tripofobia en humanos



Emilia Clarke contó la escena más caliente de Game of Thrones



Trucos para Whatsapp y avergonzar a tus contactos



Los 50 murales más famosos del mundo

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	200.121.1.131	172.16.0.122	TCP	1454	TCP segment
2	0.000011	172.16.0.122	200.121.1.131	TCP	54	TCP segment
3	0.025738	200.121.1.131	172.16.0.122	TCP	1454	TCP segment
4	0.025749	172.16.0.122	200.121.1.131	TCP	54	TCP window
5	0.070907	200.121.1.131	172.16.0.122	TCP	1454	TCP segment
6	0.070978	172.16.0.122	200.121.1.131	TCP	54	TCP dup ACK
7	0.102939	200.121.1.131	172.16.0.122	TCP	1454	TCP segment
8	0.102945	172.16.0.122	200.121.1.131	TCP	54	TCP dup ACK
9	0.132285	200.121.1.131	172.16.0.122	TCP	1454	TCP segment
10	0.120319	172.16.0.122	200.121.1.131	TCP	54	TCP dup ACK
11	0.154162	200.121.1.131	172.16.0.122	TCP	1454	TCP segment

Llega Wireshark 2.0 con nueva interfaz Qt5

```
pygame.init()
main()
```

Bien con esto hemos conseguido poner una imagen de fondo y tener una forma facil de cargar imagenes en Pygame. Aconsejo guardar esto como plantilla base de Pygame, pues en casi todos los juegos necesitaremos cargar imagenes y establecer un fondo por defecto.

DVI

4. Creando Sprites

Ahora empieza lo interesante y la esencia de los videojuegos en 2D, la creacion de Sprites.

La clase Bola

Para crear la pelota de nuestro Pong lo primero que necesitamos es la imagen de la pelota, descargar esta y guardarla como siempre en el directorio images.

Ahora vamos a crear una clase para la pelota, que sera el Sprite de nuestra pelota, un Sprite es mas que una imagen, es una superficie que puede interactuar, moverse y demas.

```
class Bola(pygame.sprite.Sprite):
    def __init__(self):
        pygame.sprite.Sprite.__init__(self)
        self.image = load_image("images/ball.png", True)
        self.rect = self.image.get_rect()
        self.rect.centerx = WIDTH / 2
        self.rect.centery = HEIGHT / 2
        self.speed = [0.5, -0.5]
```

La **línea 1** crea la clase Bola que hereda los metodos de la clase pygame.sprite.Sprite, esto es muy importante debido a que contiene metodos necesarios para el manejo de Sprites.

La **línea 2** crea el metodo init que inicializa la clase.

La **línea 3** invoca al metodo init de la clase heredada, muy importante tambien y difícil de explicar para comenzar.

La **línea 4** ya nos suena mas y lo que hace es cargar con nuestra funcion load_image() la imagen de la pelota, como vemos tenemos puesto True porque la pelota si tiene zonas transparentes.

La **línea 5** es de las cosas mas utiles de pygame la funcion self.image.get_rect() obtiene un rectangulo con las dimensiones y posicion de la imagen (en este caso self.image) y se lo asignamos a self.rect

Aqui hago un inciso para comentar que get_rect() tiene unos parametros muy utiles que podemos modificar para posicionar y redimensionar nuestra imagen, son los siguientes:

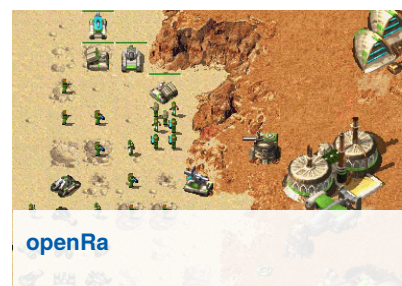
```
top, left, bottom, right
topleft, bottomleft, topright, bottomright
midtop, midleft, midbottom, midright
center, centerx, centery
size, width, height
```



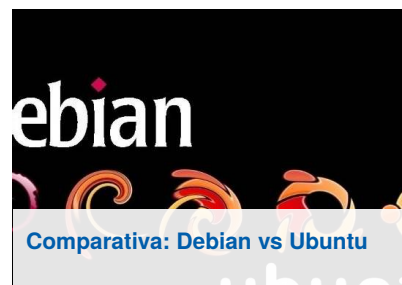
Los mejores reproductores de música para Linux



UNIX: uno de los S.O. más importantes en la historia



openRa



Comparativa: Debian vs Ubuntu



Ubuntu 16.10 vendría con Mir y Unity 8

w, h

Asi que podemos acceder a los diferentes valores como `self.rect.centerx` que nos devuelve la posicion central de la pelota respecto al ancho de la pantalla, y asi con todos es cuestion de probarlos, pero mas o menos se entiende lo que devuelve cada uno. Lo mejor de todos ellos es que si cambias el valor de alguno el resto se actualiza. Aqui lo teneis mejor explicado.

En la **línea 6 y 7** usamos las propiedades de `rect` y con `centerx` y `centery` definimos el centro de la pelota en el centro de la pantalla. (aqui vemos porque pusimos `WIDTH` y `HEIGHT` como constantes globales).

La línea 8 define la velocidad que queremos para la pelota, separamos la velocidad en dos, la velocidad en el eje x y la velocidad en el eje y, luego veremos porque.

Poniendo una Bola en nuestro juego

Bien una vez creada la clase `Bola` vamos a crear una bola, para ello como cuando cargamos el fondo, debemos cargarla antes del bucle del juego, con la siguiente línea:

```
bola = Bola()
```

Ahora solo debemos "ponerla" en nuestra ventana, ejecutando la siguiente sentencia, despues de la sentencia identica a la del fondo.

```
screen.blit(bola.image, bola.rect)
```

Como puedes ver ahora no le pasamos la imagen y en vez de unas coordenadas como en el fondo (entonces le pasamos (0, 0)) le pasamos el `rect` de la bola, con esto conseguiremos mas tarde que cada vez que movamos el `rect`, moveremos la pelota de sitio.

Es muy importante "ponerla en pantalla despues y no antes que el fondo porque si pones la pelota y luego pones el fondo encima esta no se vera, como es logico.

El programa nos queda asi:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

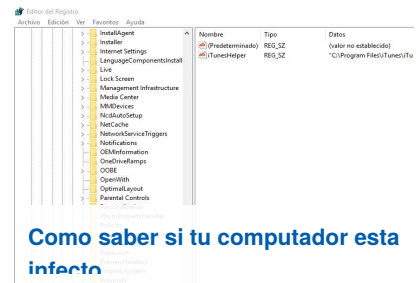
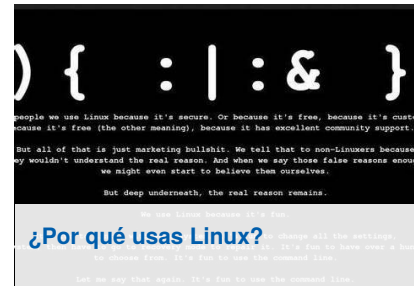
# Modulos
import sys, pygame
from pygame.locals import *

# Constantes
WIDTH = 640
HEIGHT = 480

# Clases
# -----

class Bola(pygame.sprite.Sprite):
    def __init__(self):
        pygame.sprite.Sprite.__init__(self)
        self.image = load_image("images/ball.png", True)
        self.rect = self.image.get_rect()
        self.rect.centerx = WIDTH / 2
        self.rect.centery = HEIGHT / 2
        self.speed = [0.5, -0.5]

# -----
```



```

# Funciones
# -----

def load_image(filename, transparent=False):
    try: image = pygame.image.load(filename)
    except pygame.error, message:
        raise SystemExit, message
    image = image.convert()
    if transparent:
        color = image.get_at((0,0))
        image.set_colorkey(color, RLEACCEL)
    return image

# -----

def main():
    screen = pygame.display.set_mode((WIDTH, HEIGHT))
    pygame.display.set_caption("Pruebas Pygame")

    background_image = load_image('images/fondo_pong.png')
    bola = Bola()

    while True:
        for eventos in pygame.event.get():
            if eventos.type == QUIT:
                sys.exit(0)

        screen.blit(background_image, (0, 0))
        screen.blit(bola.image, bola.rect)
        pygame.display.flip()
    return 0

if __name__ == '__main__':
    pygame.init()
    main()

```

Ahora tenemos una pelota en el centro de nuestra pantalla lo siguiente es conseguir que se mueva e interactue con los bordes.



5. Moviendo Sprites

El metodo actualizar

A continuacion aprenderemos a mover nuestra bola por la superficie y a calcular los rebotes, para ello emplearemos matematicas y fisica basica.

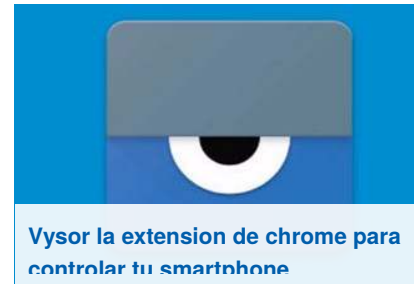
```

def actualizar(self, time):
    self.rect.centerx += self.speed[0] * time

```



3.000 servidores web afectados por ransomware para Linux



Vysor la extension de chrome para controlar tu smartphone



Elive Topaz, una distribución que tenés que probar!!!



Salió la beta de Linux Mint 17.3 Rosa



Las 11 Mejores distribuciones de Linux

```

self.rect.centery += self.speed[1] * time
if self.rect.left <= 0 or self.rect.right >= WIDTH:
    self.speed[0] = -self.speed[0]
    self.rect.centerx += self.speed[0] * time
if self.rect.top <= 0 or self.rect.bottom >= HEIGHT:
    self.speed[1] = -self.speed[1]
    self.rect.centery += self.speed[1] * time

```

La **línea 1** define el metodo, recibe el parametro self (como siempre) y el parametro time que es el tiempo transcurrido, mas adelante lo explicamos.

La **línea 2 y 3** usa la fisica basica de espacio es igual a la velocidad por el tiempo ($e = v \cdot t$), por tanto establecemos que el centro de nuestro rectangulo en x es el valor que tenia (self.rect.centerx) mas (+) la velocidad a la que se mueve en el eje x (self.speed[0]) por (*) el tiempo transcurrido (time). Lo mismo se aplica al eje y en la linea 3.

La **línea 4, 5 y 6** establece que si la parte izquierda del rectangulo de la bola es menor o igual a 0 o mayor o igual a el ancho de la pantalla (WIDTH), es decir, que este en el extremo izquierdo o derecho, la velocidad de x (self.speed[0]) cambie de signo (-self.speed[0]) con esto conseguiremos que vaya hacia el otro lado.

Las **líneas 7, 8 y 9** es lo mismo pero en el eje y como se puede ver.

Creando un reloj

Ahora vamos a crear un reloj que controle el tiempo del juego, esto es importante para el movimiento, pues sabemos cuanto tiempo a pasado desde la ultima actualizacion de la pelota y con ello poder situarla en el espacio.

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

Esta linea va justo antes de entrar en el bucle del juego y sirve para crear el reloj con el que gestionar el tiempo.

Ahora necesitamos saber cuanto tiempo pasa cada vez que se ejecuta una interseccion del bucle, para ello dentro del bucle ponemos como primera linea:

```
time = clock.tick(60)
```

El 60 que se le pasa como parametro es el framerate, con el nos aseguramos de que el juego no va a mas de la velocidad estipulada. Con ello conseguimos que el juego corra igual en todos los ordenadores.

Por ultimo debemos actualizar la posicion de la bola antes de actualizarla en la ventana, es decir antes de los screen.blit.

```
bola.actualizar(time)
```

Como ves le pasamos el parametro time que es el tiempo que ha pasado desde la ultima vez que se ejecuto.

El juego queda asi:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
```

```
# Modulos
import sys, pygame
from pygame.locals import *
```

```
# Constantes
WIDTH = 640
HEIGHT = 480
```



Gentoo, instalar entorno gráfico y Xfce



USB Killer 2.0, el pendrive que "mata" ordenadores



6 consejos para prevenir el espionaje por Edward Snowden



Distribuciones GNU/Linux para Hacking y Seguridad



Ubuntu AIO: todos los sabores Ubuntu (x86 - x64)

```

# Clases
# -----

class Bola(pygame.sprite.Sprite):
    def __init__(self):
        pygame.sprite.Sprite.__init__(self)
        self.image = load_image("images/ball.png", True)
        self.rect = self.image.get_rect()
        self.rect.centerx = WIDTH / 2
        self.rect.centery = HEIGHT / 2
        self.speed = [0.5, -0.5]

    def actualizar(self, time):
        self.rect.centerx += self.speed[0] * time
        self.rect.centery += self.speed[1] * time
        if self.rect.left <= 0 or self.rect.right >= WIDTH:
            self.speed[0] = -self.speed[0]
            self.rect.centerx += self.speed[0] * time
        if self.rect.top <= 0 or self.rect.bottom >= HEIGHT:
            self.speed[1] = -self.speed[1]
            self.rect.centery += self.speed[1] * time

# -----

# Funciones
# -----

def load_image(filename, transparent=False):
    try: image = pygame.image.load(filename)
    except pygame.error, message:
        raise SystemExit, message
    image = image.convert()
    if transparent:
        color = image.get_at((0,0))
        image.set_colorkey(color, RLEACCEL)
    return image

# -----

def main():
    screen = pygame.display.set_mode((WIDTH, HEIGHT))
    pygame.display.set_caption("Pruebas Pygame")

    background_image = load_image("images/fondo_pong.png")
    bola = Bola()

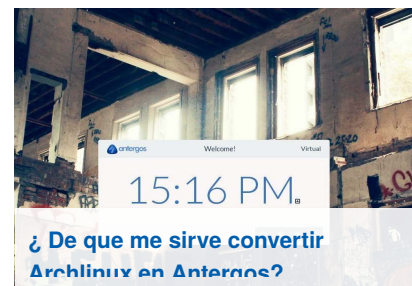
    clock = pygame.time.Clock()

    while True:
        time = clock.tick(60)
        for eventos in pygame.event.get():
            if eventos.type == QUIT:
                sys.exit(0)

        bola.actualizar(time)
        screen.blit(background_image, (0, 0))
        screen.blit(bola.image, bola.rect)
        pygame.display.flip()
    return 0

if __name__ == '__main__':
    pygame.init()

```




```
main()
```

Ya nuestra pelota se mueve y rebota contra las paredes. No es la panacea del movimiento, pero esta bien.

```
elif
```

6. Control del teclado

Bien ya tenemos nuestra pelota que se mueve como loca por toda la pantalla. El siguiente paso es crear un nuevo Sprite para las palas, este a diferencia de la bola no se movera solo sino que lo controlaremos nosotros por el teclado.

Creando el Sprite Pala

La clase Pala es casi identica a la clase Bola, salvo por unos pequeños cambios:

```
class Pala(pygame.sprite.Sprite):
    def __init__(self, x):
        pygame.sprite.Sprite.__init__(self)
        self.image = load_image("images/pala.png")
        self.rect = self.image.get_rect()
        self.rect.centerx = x
        self.rect.centery = HEIGHT / 2
        self.speed = 0.5
```

Como vemos es identica salvo que ahora le pasamos el parametro x para usarlo en self.rect.centerx, esto es debido a que necesitamos dos palas una en la parte izquierda y otra en la derecha, con el parametro x definimos a que altura del eje x queremos colocar el Sprite.

Otro cambio es la velocidad, como la pala del Pong solo se mueve en el eje y no definimos velocidad para el eje x.

para crear la pala del jugador ya sabeis, debajo de donde habeis creado la bola poneis la siguiente linea:

```
pala_jug = Pala(30)
```

La llamo pala_jug porque habra otra que sera la pala_cpu que sera la que maneje el ordenador. Como ves le paso como valor de x = 30, esto quiere decir que centerx estara a 30 px del borde derecho de la ventana.

Por ultimo dentro del bucle lo de siempre, despues de actualizar el fondo y la bola actualizamos la pala:

```
screen.blit(pala_jug.image, pala_jug.rect)
```

Moviendo la pala

Para mover la pala definimos el metodo mover dentro de la clase Pala:

```
def mover(self, time, keys):
    if self.rect.top >= 0:
        if keys[K_UP]:
            self.rect.centery -= self.speed * time
    if self.rect.bottom <= HEIGHT:
        if keys[K_DOWN]:
            self.rect.centery += self.speed * time
```

Recibe los parametros self y time como el **metodo** actualizar de la bola y ademas



Canonical va ganando la batalla por la convergencia



7 cosas a hacer después de instalar Ubuntu 15.10



Linus Torvalds: la seguridad perfecta en el código abierto



SteamOS aún lejos de aproximarse al rendimiento de



Deshabilita Hello, Pocket y Reader en Firefox de un plumazo

recibe el parametro keys que luego definiremos y que es una lista con el valor booleano de las teclas pulsadas.

La **línea 2 y 5** comprueban que la parte superior (en el caso de la línea 2) de la pala sea mayor o igual a 0 y que la parte inferior de la pala (**línea 5**) sea menor o igual que la altura de la ventana.

Resumiendo comprueban que la pala no se sale de la ventana.

La **línea 3** comprueba si la constante K_UP de keys es 1, lo que querria decir que tenemos presionada la tecla de la flecha hacia arriba del teclado.

La **línea 5** en caso de tener la tecla presionada disminuye el valor de centery haciendo que la pala se mueva hacia arriba.

La **línea 6 y 7** hacen lo mismo, pero para abajo y aumentando el valor de centery.

Ahora solo debemos saber que teclas se estan pulsando creando la variable keys, esto se consigue añadiendo la siguiente linea en el bucle principal, justo despues de comprobar el tiempo transcurrido con time.

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

Esto nos devuelve las teclas pulsadas en una lista como explicamos arriba.

Por ultimo debemos llamar al metodo mover en el bucle justo despues de actualizar la bola, con la linea:

```
pala_jug.mover(time, keys)
```

El codigo nos queda asi:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

# Modulos
import sys, pygame
from pygame.locals import *

# Constantes
WIDTH = 640
HEIGHT = 480

# Clases
# -----

class Bola(pygame.sprite.Sprite):
    def __init__(self):
        pygame.sprite.Sprite.__init__(self)
        self.image = load_image("images/ball.png", True)
        self.rect = self.image.get_rect()
        self.rect.centerx = WIDTH / 2
        self.rect.centery = HEIGHT / 2
        self.speed = [0.5, -0.5]

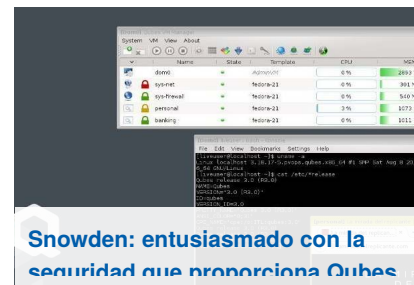
    def actualizar(self, time):
        self.rect.centerx += self.speed[0] * time
        self.rect.centery += self.speed[1] * time
        if self.rect.left <= 0 or self.rect.right >= WIDTH:
            self.speed[0] = -self.speed[0]
            self.rect.centerx += self.speed[0] * time
        if self.rect.top <= 0 or self.rect.bottom >= HEIGHT:
```



Lo mejor de Android para tablets



Visualiza el tráfico de la red Tor con TorFlow



Snowden: entusiasmado con la seguridad que proporciona Qubes



```

        self.speed[1] = -self.speed[1]
        self.rect.centery += self.speed[1] * time

class Pala(pygame.sprite.Sprite):
    def __init__(self, x):
        pygame.sprite.Sprite.__init__(self)
        self.image = load_image("images/pala.png")
        self.rect = self.image.get_rect()
        self.rect.centerx = x
        self.rect.centery = HEIGHT / 2
        self.speed = 0.5

    def mover(self, time, keys):
        if self.rect.top >= 0:
            if keys[K_UP]:
                self.rect.centery -= self.speed * time
        if self.rect.bottom <= HEIGHT:
            if keys[K_DOWN]:
                self.rect.centery += self.speed * time

# -----

# Funciones
# -----

def load_image(filename, transparent=False):
    try: image = pygame.image.load(filename)
    except pygame.error, message:
        raise SystemExit, message
    image = image.convert()
    if transparent:
        color = image.get_at((0,0))
        image.set_colorkey(color, RLEACCEL)
    return image

# -----

def main():
    screen = pygame.display.set_mode((WIDTH, HEIGHT))
    pygame.display.set_caption("Pruebas Pygame")

    background_image = load_image("images/fondo_pong.png")
    bola = Bola()
    pala_jug = Pala(30)

    clock = pygame.time.Clock()

    while True:
        time = clock.tick(60)
        keys = pygame.key.get_pressed()
        for eventos in pygame.event.get():
            if eventos.type == QUIT:
                sys.exit(0)

        bola.actualizar(time)
        pala_jug.mover(time, keys)
        screen.blit(background_image, (0, 0))
        screen.blit(bola.image, bola.rect)
        screen.blit(pala_jug.image, pala_jug.rect)
        pygame.display.flip()
    return 0

if __name__ == '__main__':

```



```
pygame.init()
main()
```

Ya tenemos nuestra pala que podemos controlar con el teclado, pero la bola como vemos para olimpicamente de ella y la atraviesa. A continuacion aprenderemos a crear colisiones entre Sprites

Lir

7. Colisiones

Tenemos una pelota que se mueve como loca y una pala que va para arriba y para abajo a nuestra voluntad, es hora de que se den cuenta que estan en el mismo mundo. Vamos a crear colisiones entre los dos sprites.

Actualizando el metodo actualizar de la clase Bola

Saber si un Sprite colisiona con otro es muy facil en Python, basta con ejecutar el siguiente metodo:

```
pygame.sprite.collide_rect(objeto1, objeto2)
```

Esto comprueba si el rectangulo del Sprite objeto1 esta en contacto con el rectangulo de objeto2, por lo que es tan facil como mejorar el metodo actualizar de la clase Bola para que lo tenga en cuenta:

```
def actualizar(self, time, pala_jug):
    self.rect.centerx += self.speed[0] * time
    self.rect.centery += self.speed[1] * time
    if self.rect.left <= 0 or self.rect.right >= WIDTH:
        self.speed[0] = -self.speed[0]
        self.rect.centerx += self.speed[0] * time
    if self.rect.top <= 0 or self.rect.bottom >= HEIGHT:
        self.speed[1] = -self.speed[1]
        self.rect.centery += self.speed[1] * time

    if pygame.sprite.collide_rect(self, pala_jug):
        self.speed[0] = -self.speed[0]
        self.rect.centerx += self.speed[0] * time
```

Como vemos todo igual, salvo que ahora recibe un parametro mas que es **pala_jug** por el que pasaremos el Sprite con el que queremos comprobar si colisiona, en este caso **pala_jug**.

Luego al final añadimos 3 líneas con un nuevo condicional con el que comprobamos si la pelota choca contra la pala, en caso afirmativo cambiamos la direccion de la bola como cuando choca con el borde izquierdo de la ventana.

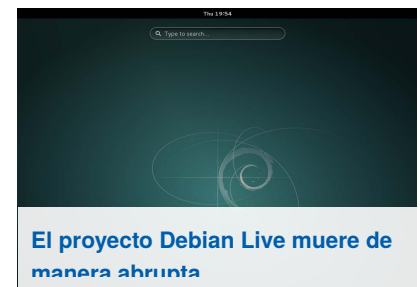
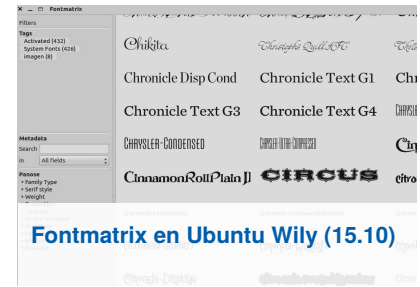
Por ultimo solo actualizamos la llamada a la funcion actualizar que hacemos desde el bucle para pasarle la **pala_jug**.

```
bola.actualizar(time, pala_jug)
```

Y con esto ya hemos logrado que nuestra bola detecte si ha chocado contra la pala del jugador en caso afirmativo "rebota".

Asi nos queda el codigo:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
```




```

# Modulos
import sys, pygame
from pygame.locals import *

# Constantes
WIDTH = 640
HEIGHT = 480

# Clases
# -----

class Bola(pygame.sprite.Sprite):
    def __init__(self):
        pygame.sprite.Sprite.__init__(self)
        self.image = load_image("images/ball.png", True)
        self.rect = self.image.get_rect()
        self.rect.centerx = WIDTH / 2
        self.rect.centery = HEIGHT / 2
        self.speed = [0.5, -0.5]

    def actualizar(self, time, pala_jug):
        self.rect.centerx += self.speed[0] * time
        self.rect.centery += self.speed[1] * time
        if self.rect.left <= 0 or self.rect.right >= WIDTH:
            self.speed[0] = -self.speed[0]
        self.rect.centerx += self.speed[0] * time
        if self.rect.top <= 0 or self.rect.bottom >= HEIGHT:
            self.speed[1] = -self.speed[1]
        self.rect.centery += self.speed[1] * time

        if pygame.sprite.collide_rect(self, pala_jug):
            self.speed[0] = -self.speed[0]
            self.rect.centerx += self.speed[0] * time

class Pala(pygame.sprite.Sprite):
    def __init__(self, x):
        pygame.sprite.Sprite.__init__(self)
        self.image = load_image("images/pala.png")
        self.rect = self.image.get_rect()
        self.rect.centerx = x
        self.rect.centery = HEIGHT / 2
        self.speed = 0.5

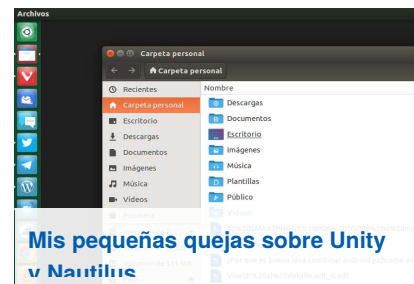
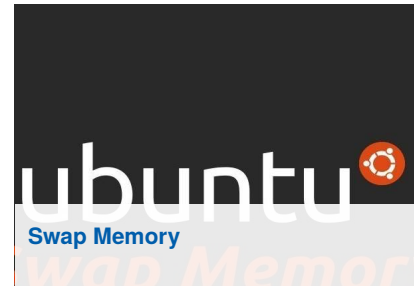
    def mover(self, time, keys):
        if self.rect.top >= 0:
            if keys[K_UP]:
                self.rect.centery -= self.speed * time
        if self.rect.bottom <= HEIGHT:
            if keys[K_DOWN]:
                self.rect.centery += self.speed * time

# -----

# Funciones
# -----

def load_image(filename, transparent=False):
    try: image = pygame.image.load(filename)
    except pygame.error, message:
        raise SystemExit, message
    image = image.convert()
    if transparent:
        color = image.get_at((0,0))

```



```

        image.set_colorkey(color, RLEACCEL)
        return image

# -----

def main():
    screen = pygame.display.set_mode((WIDTH, HEIGHT))
    pygame.display.set_caption("Pruebas Pygame")

    background_image = load_image('images/fondo_pong.png')
    bola = Bola()
    pala_jug = Pala(30)

    clock = pygame.time.Clock()

    while True:
        time = clock.tick(60)
        keys = pygame.key.get_pressed()
        for eventos in pygame.event.get():
            if eventos.type == QUIT:
                sys.exit(0)

        bola.actualizar(time, pala_jug)
        pala_jug.mover(time, keys)
        screen.blit(background_image, (0, 0))
        screen.blit(bola.image, bola.rect)
        screen.blit(pala_jug.image, pala_jug.rect)
        pygame.display.flip()
    return 0

if __name__ == '__main__':
    pygame.init()
    main()

```

En el siguiente paso crearemos la pala del jugador de la CPU y la dotaremos de Inteligencia Artificial para que golpee la pelota.



8. Inteligencia Artificial

En el siguiente tema no introduciremos ningún concepto nuevo de Pygame, pero si usaremos un poco de todo lo usado en los anteriores tutoriales para crear la pala que controlara el ordenador, se podría decir que es una Inteligencia Artificial muy básica.

Creando la pala de la CPU

Esto se hace igual que como creamos la **pala_jug** o la bola, ponemos la siguiente línea en la función principal debajo de **pala_jug**.



El Proyecto Debian solicita ayuda con las traducciones



Audacious 3.7 añade soporte para grabación en streaming



linux en modo persistencia



Actualización Flash 548 (Linux)
para Flash Player 11.2 para Linux.



Chrome OS se fusionará con Android en 2017

```
pala_cpu = Pala(WIDTH - 30)
```

En su momento `pala_jug` le pasamos como parametro el valor 30 que queria decir que el `centerx` estaba a 30 pixeles de el borde izquierdo, ahora le pasamos `WIDTH - 30`, es decir a 30 pixeles del borde derecho.

Tambien debemos añadir la linea:

```
screen.blit(pala_cpu.image, pala_cpu.rect)
```

En el bucle principal, debajo de los demas blit, como siempre. Con esto ya tenemos nuestro Sprite puesto en pantalla.

Ahora vamos a hacer que detecte tambien las colisiones, para ello volvemos a actualizar el metodo actualizar de la clase Bola:

```
def actualizar(self, time, pala_jug, pala_cpu):
    self.rect.centerx += self.speed[0] * time
    self.rect.centery += self.speed[1] * time
    if self.rect.left <= 0 or self.rect.right >= WIDTH:
        self.speed[0] = -self.speed[0]
        self.rect.centerx += self.speed[0] * time
    if self.rect.top <= 0 or self.rect.bottom >= HEIGHT:
        self.speed[1] = -self.speed[1]
        self.rect.centery += self.speed[1] * time

    if pygame.sprite.collide_rect(self, pala_jug):
        self.speed[0] = -self.speed[0]
        self.rect.centerx += self.speed[0] * time

    if pygame.sprite.collide_rect(self, pala_cpu):
        self.speed[0] = -self.speed[0]
        self.rect.centerx += self.speed[0] * time
```

Como podemos ver añadimos otro parametro al metodo **pala_cpu** que servira para añadirle el Sprite **pala_cpu** como hicimos con **pala_jug** y añadimos las tres ultimas lineas que son identicas a las anteriores salvo que ahora para **pala_cpu**.

Recordad pasarle el parametro nuevo a la linea que llama a la funcion actualizar de la bola en el bucle del juego. Así:

```
ball.update(time, pala_jug, pala_cpu)
```

Dotando de Inteligencia Artificial a la Pala

Bueno ahora viene lo interesante, lograr que la pala se mueva para golpear la bola, esto se hace definiendo otro metodo en la clase Pala al metodo lo llamare `ia`.

```
def ia(self, time, ball):
    if ball.speed[0] >= 0 and ball.rect.centerx >= WIDTH/2:
        if self.rect.centery < ball.rect.centery:
            self.rect.centery += self.speed * time
        if self.rect.centery > ball.rect.centery:
            self.rect.centery -= self.speed * time
```

En la **línea 1** vemos que recibe como siempre `self` y `time` y aparte recibe `ball` que es la bola, es necesario pues el metodo necesita conocer donde esta la bola.

En la **línea 2** comprobamos que `ball.speed[0] >= 0`, es decir, que la velocidad en el eje x de la pelota sea positiva, es decir, que la pelota se este moviendo hacia la derecha (hacia la pala de la cpu) y tambien comprueba que `ball.rect.centerx >= WIDTH/2` es decir que el centro x de la pelota sea mayor o igual que el centro del



An Example Model
For a Linux OS
(Not Authoritative: Many possible models)

Linux Kernel

Applications/Services

¿Sabes lo que es un Nucleo o Kernel ?



Reparar sectores dañados del disco duro con Ubuntu



Una introducción a la complicada Firewall



Thermal, un demonio que te mantiene fresco

tablero, es decir, que la pelota este en el campo de la cpu.

Por tanto la **línea 2** es un condicional que comprueba que la pelota vaya hacia donde esta la pala de la cpu y que este en su campo, sino, que no se mueva. Esto se hace para que la CPU no sea invencible y no llegue a todas las pelotas.

La **línea 3** comprueba si el centery de la pelota es menor que el centery de la bola, es decir si la pala esta mas arriba que que la pelota en cullo caso, ejecuta la **línea 4** que mueve la pala de la cpu hacia abajo.

Las **líneas 5 y 6** hacen lo mismo, pero a la inversa como se ve a simple vista.

Ahora solo nos queda llamar a la ia junto con las llamadas actualizar de la bola y mover de la pala_cpu.

```
pala_cpu.ia(time, bola)
```

El codigo nos queda de la siguiente manera:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

# Modulos
import sys, pygame
from pygame.locals import *

# Constantes
WIDTH = 640
HEIGHT = 480

# Clases
# -----

class Bola(pygame.sprite.Sprite):
    def __init__(self):
        pygame.sprite.Sprite.__init__(self)
        self.image = load_image("images/ball.png", True)
        self.rect = self.image.get_rect()
        self.rect.centerx = WIDTH / 2
        self.rect.centery = HEIGHT / 2
        self.speed = [0.5, -0.5]

    def actualizar(self, time, pala_jug, pala_cpu):
        self.rect.centerx += self.speed[0] * time
        self.rect.centery += self.speed[1] * time
        if self.rect.left <= 0 or self.rect.right >= WIDTH:
            self.speed[0] = -self.speed[0]
            self.rect.centerx += self.speed[0] * time
        if self.rect.top <= 0 or self.rect.bottom >= HEIGHT:
            self.speed[1] = -self.speed[1]
            self.rect.centery += self.speed[1] * time

        if pygame.sprite.collide_rect(self, pala_jug):
            self.speed[0] = -self.speed[0]
            self.rect.centerx += self.speed[0] * time

        if pygame.sprite.collide_rect(self, pala_cpu):
            self.speed[0] = -self.speed[0]
            self.rect.centerx += self.speed[0] * time

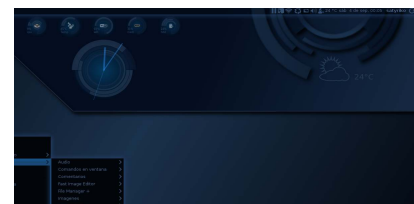
class Pala(pygame.sprite.Sprite):
    def __init__(self, x):
        pygame.sprite.Sprite.__init__(self)
```



NetSurf: un navegador que apenas consume memoria RAM



Llega el primer ransomware que afecta a Linux



Archlinux, Instalacion Basica del Sistema Operativo



El futuro de Mozilla Firefox no gusta a la comunidad



Manjaro i3 15.11 liberado


```

self.image = load_image("images/pala.png")
self.rect = self.image.get_rect()
self.rect.centerx = x
self.rect.centery = HEIGHT / 2
self.speed = 0.5

def mover(self, time, keys):
    if self.rect.top >= 0:
        if keys[K_UP]:
            self.rect.centery -= self.speed * time
    if self.rect.bottom <= HEIGHT:
        if keys[K_DOWN]:
            self.rect.centery += self.speed * time

def ia(self, time, ball):
    if ball.speed[0] >= 0 and ball.rect.centerx >= WIDTH/2:
        if self.rect.centery < ball.rect.centery:
            self.rect.centery += self.speed * time
        if self.rect.centery > ball.rect.centery:
            self.rect.centery -= self.speed * time

# -----

# Funciones
# -----

def load_image(filename, transparent=False):
    try: image = pygame.image.load(filename)
    except pygame.error, message:
        raise SystemExit, message
    image = image.convert()
    if transparent:
        color = image.get_at((0,0))
        image.set_colorkey(color, RLEACCEL)
    return image

# -----

def main():
    screen = pygame.display.set_mode((WIDTH, HEIGHT))
    pygame.display.set_caption("Pruebas Pygame")

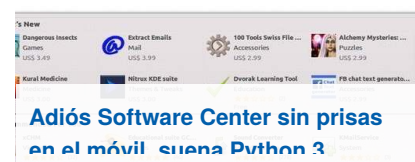
    background_image = load_image("images/fondo_pong.png")
    bola = Bola()
    pala_jug = Pala(30)
    pala_cpu = Pala(WIDTH - 30)

    clock = pygame.time.Clock()

    while True:
        time = clock.tick(60)
        keys = pygame.key.get_pressed()
        for eventos in pygame.event.get():
            if eventos.type == QUIT:
                sys.exit(0)

        bola.actualizar(time, pala_jug, pala_cpu)
        pala_jug.mover(time, keys)
        pala_cpu.ia(time, bola)
        screen.blit(background_image, (0, 0))
        screen.blit(bola.image, bola.rect)
        screen.blit(pala_jug.image, pala_jug.rect)
        screen.blit(pala_cpu.image, pala_cpu.rect)

```



```
pygame.display.flip()
return 0

if __name__ == '__main__':
    pygame.init()
    main()
```

Ahora el juego es jugable, pero no tiene ni sistema de puntuacion, ni sonidos ni nada de nada. Pronto lo solucionaremos.

Co

9. Sistema de puntuacion

En los dos siguientes temas vamos a aprender a manejar fuentes tipograficas en Pygame. Crearemos un sistema de puntuacion para nuestro Pong, consistira en dos marcadores para cada jugador que si consigues meter un punto al rival aumenta en uno. ¿Sencillo no? Vamos alla.

Creando un sistema de puntuacion

Lo primero es crear un sistema que controle los puntos, es decir, comprobar si la bola traspasa las palas y toca el borde de la ventana que tiene detras. Esto lo controlara como siempre la bola y su metodo actualizar.

```
def update(self, time, pala_jug, pala_cpu, puntos):
    self.rect.centerx += self.speed[0] * time
    self.rect.centery += self.speed[1] * time

    if self.rect.left <= 0:
        puntos[1] += 1
    if self.rect.right >= WIDTH:
        puntos[0] += 1

    if self.rect.left <= 0 or self.rect.right >= WIDTH:
        self.speed[0] = -self.speed[0]
        self.rect.centerx += self.speed[0] * time
    if self.rect.top <= 0 or self.rect.bottom >= HEIGHT:
        self.speed[1] = -self.speed[1]
        self.rect.centery += self.speed[1] * time

    if pygame.sprite.collide_rect(self, pala_jug):
        self.speed[0] = -self.speed[0]
        self.rect.centerx += self.speed[0] * time

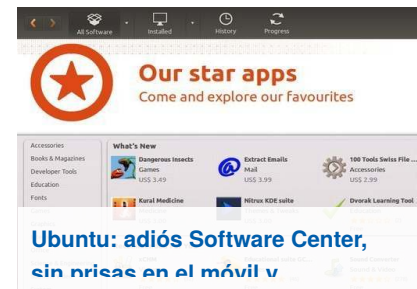
    if pygame.sprite.collide_rect(self, pala_cpu):
        self.speed[0] = -self.speed[0]
        self.rect.centerx += self.speed[0] * time

    return puntos
```

Como vemos le añadimos un nuevo parametro llamado puntos, puntos es una lista que contiene los puntos de los dos jugadores en el **puntos[0]** los puntos del jugador y en el **puntos[1]** los puntos de la cpu.

Luego añadimos las **lineas de la 5 a la 8** que controlan si la parte izquierda de la pelota (**línea 5**) toca el el borde izquierdo de la ventana en cuyo caso aumenta **puntos[1]** (el marcador de la cpu) en 1 (**línea 6**). La **líneas 7 y 8** hacen lo mismo, pero a la inversa.

Por ultimo al final del metodo se retorna puntos, necesario para almacenarla en



Tester v2.1.13
Engine: POSIX AIO - Buffered: No - Direct: Yes - Block Size: 4096 - Disk Target: 4 - Result: MB/s

	Min	Avg	Max
Btrfs	41.4	41.7	42.0
EXT4	99.8	100.6	101.9
XFS	107.8	109.0	111.5
ZFS	106.1	107.4	109.4

48% More is Better



una variable.

Ahora debemos crear la lista puntos en nuestra funcion principal, yo la he creado justo antes de entrar en el bucle del juego:

```
puntos = [0, 0]
```

Ahora tenemos que modificar la llamada a **bola.actualizar** dentro del bucle del juego para pasarle nuestra lista puntos y recuperarla de nuevo con los posibles nuevos valores.

```
puntos = bola.actualizar(time, pala_jug, pala_cpu, puntos)
```

Por lo que el codigo nos queda asi:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

# Modulos
import sys, pygame
from pygame.locals import *

# Constantes
WIDTH = 640
HEIGHT = 480

# Clases
# -----

class Bola(pygame.sprite.Sprite):
    def __init__(self):
        pygame.sprite.Sprite.__init__(self)
        self.image = load_image("images/ball.png", True)
        self.rect = self.image.get_rect()
        self.rect.centerx = WIDTH / 2
        self.rect.centery = HEIGHT / 2
        self.speed = [0.5, -0.5]

    def actualizar(self, time, pala_jug, pala_cpu, puntos):
        self.rect.centerx += self.speed[0] * time
        self.rect.centery += self.speed[1] * time

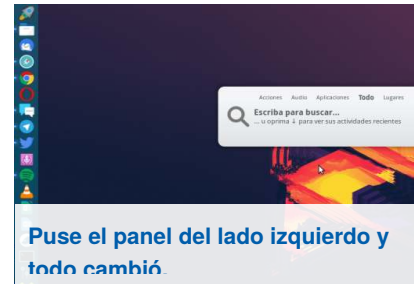
        if self.rect.left <= 0:
            puntos[1] += 1
        if self.rect.right >= WIDTH:
            puntos[0] += 1

        if self.rect.left <= 0 or self.rect.right >= WIDTH:
            self.speed[0] = -self.speed[0]
            self.rect.centerx += self.speed[0] * time
        if self.rect.top <= 0 or self.rect.bottom >= HEIGHT:
            self.speed[1] = -self.speed[1]
            self.rect.centery += self.speed[1] * time

        if pygame.sprite.collide_rect(self, pala_jug):
            self.speed[0] = -self.speed[0]
            self.rect.centerx += self.speed[0] * time

        if pygame.sprite.collide_rect(self, pala_cpu):
            self.speed[0] = -self.speed[0]
            self.rect.centerx += self.speed[0] * time

    return puntos
```



```

class Pala(pygame.sprite.Sprite):
    def __init__(self, x):
        pygame.sprite.Sprite.__init__(self)
        self.image = load_image("images/pala.png")
        self.rect = self.image.get_rect()
        self.rect.centerx = x
        self.rect.centery = HEIGHT / 2
        self.speed = 0.5

    def mover(self, time, keys):
        if self.rect.top >= 0:
            if keys[K_UP]:
                self.rect.centery -= self.speed * time
        if self.rect.bottom <= HEIGHT:
            if keys[K_DOWN]:
                self.rect.centery += self.speed * time

    def ia(self, time, ball):
        if ball.speed[0] >= 0 and ball.rect.centerx >= WIDTH/2:
            if self.rect.centery < ball.rect.centery:
                self.rect.centery += self.speed * time
            if self.rect.centery > ball.rect.centery:
                self.rect.centery -= self.speed * time

# -----

# Funciones
# -----

def load_image(filename, transparent=False):
    try: image = pygame.image.load(filename)
    except pygame.error, message:
        raise SystemExit, message
    image = image.convert()
    if transparent:
        color = image.get_at((0,0))
        image.set_colorkey(color, RLEACCEL)
    return image

# -----

def main():
    screen = pygame.display.set_mode((WIDTH, HEIGHT))
    pygame.display.set_caption("Pruebas Pygame")

    background_image = load_image("images/fondo_pong.png")
    bola = Bola()
    pala_jug = Pala(30)
    pala_cpu = Pala(WIDTH - 30)

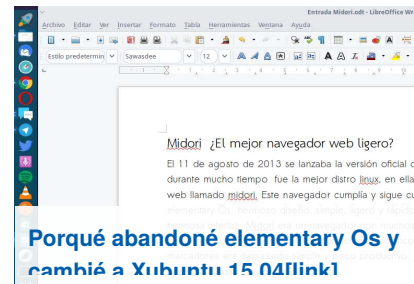
    clock = pygame.time.Clock()

    puntos = [0, 0]

    while True:
        time = clock.tick(60)
        keys = pygame.key.get_pressed()
        for eventos in pygame.event.get():
            if eventos.type == QUIT:
                sys.exit(0)

        puntos = bola.actualizar(time, pala_jug, pala_cpu, puntos)

```




```

    pala_jug.mover(time, keys)
    pala_cpu.ia(time, bola)
    screen.blit(background_image, (0, 0))
    screen.blit(bola.image, bola.rect)
    screen.blit(pala_jug.image, pala_jug.rect)
    screen.blit(pala_cpu.image, pala_cpu.rect)
    pygame.display.flip()
    return 0

if __name__ == '__main__':
    pygame.init()
    main()

```

Pr

Apendice: contenido propio

Bien, hasta aca fue el tutorial de iniciacion de Loser Juegos, pero este deja pendiente el mostrar el marcador en el juego, asi que va un pequeño apendice con la solucion que encuentre para esto.

Tambien agrego como hacer para que cada vez que se anota un tanto, la pelota vuelva a salir del centro, y como hacer para quitar el juego presionando la una tecla, en este caso, la **q**.

Tanteador en pantalla

Lo primero que tenemos que hacer es crear una clase Text que use el modulo font de pygame.

De todas formas formas, antes de ir a esta clase, voy a declarar una constanste para declarar el color blanco que utilizaremos para el texto mas adelante:

```
white = (255,255,255)
```

De esta forma, las primeras lineas del programa ahora nos quedan asi:

```

#!/usr/bin/env python
# -*- coding: utf-8 -*-

# Modulos
import sys, pygame
from pygame.locals import *

# Constantes
WIDTH = 640
HEIGHT = 480
white = (255,255,255)

```

Ahora si, en la seccion de clases de nuestro codigo creamos la clase Text:

```

class Text (pygame.font.Font):
    def __init__(self, FontName = None, FontSize = 30):
        pygame.font.init()
        self.font = pygame.font.Font(FontName, FontSize)
        self.size = FontSize

    def render (self, surface, text, color, pos):
        text = unicode(text, "UTF-8")
        x, y = pos
        for i in text.split("\n"):

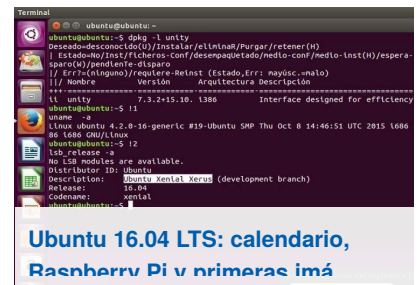
```



Arch - Gnome Screenshot
 (Incluir puntero)
 (Capturas en .png)
 (Incluir el borde de ventana)



**Qué hacer después de instalar
 Ubuntu 15.10!!!**



```
surface.blit(self.font.render(i, 1, color), (x, y))
y += self.size
```

y antes de ingresar al bucle del juego instanciamos la clase Text:

```
text = Text()
```

y a la hora de actualizar la pantalla agregamos la orden para que se actualice tambien el resultado en el marcador:

```
numero1=str(puntos[1])
text.render(screen, "P1: "+numero1, white, (20, 0))
numero0=str(puntos[0])
```

Salir con una tecla

Este paso es bastante sencillo, simplemente basta con agregar al metodo **mover** de la clase **Pala** las siguientes lineas:

```
if keys[K_q]:
    sys.exit(0)
```

Reponiendo la bola del medio

Para lograr que luego de cada punto la bola salga del medio, simplemente editamos el metodo **actualizar** de la clase **Bola**, y modificamos las lineas en que le decimo que agregue un punto para que reubique la bola en el centro:

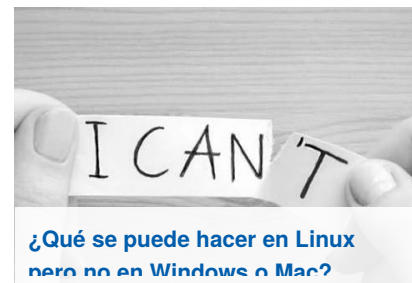
```
if self.rect.left <= 0:
    puntos[0] += 1
    self.rect.centerx = WIDTH / 2
    self.rect.centery = HEIGHT / 2
    self.speed = [-0.5, 0.5]
if self.rect.right >= WIDTH:
    puntos[1] += 1
    self.rect.centerx = WIDTH / 2
    self.rect.centery = HEIGHT / 2
    self.speed = [0.5, -0.5]
```

Finalmente, el codigo completo con las modificaciones mias quedaria asi:

```
import pygame
from pygame.locals import *
import sys, os
if not pygame.font: print 'Warning, fonts disabled'
# Constantes
WIDTH = 640
HEIGHT = 480
white = (255,255,255)

# Clases
class Bola(pygame.sprite.Sprite, pygame.font.Font):
    def __init__(self):
        pygame.sprite.Sprite.__init__(self)
        self.image = load_image("img/ball.png", True)
        self.rect = self.image.get_rect()
        self.rect.centerx = WIDTH / 2
        self.rect.centery = HEIGHT / 2
        self.speed = [0.5, -0.5]

    def actualizar(self, time, pala_jug, pala_cpu, puntos):
        self.rect.centerx += self.speed[0] * time
        self.rect.centery += self.speed[1] * time
```



```

if self.rect.left <= 0:
    puntos[0] += 1
    self.rect.centerx = WIDTH / 2
    self.rect.centery = HEIGHT / 2
    self.speed = [-0.5, 0.5]
if self.rect.right >= WIDTH:
    puntos[1] += 1
    self.rect.centerx = WIDTH / 2
    self.rect.centery = HEIGHT / 2
    self.speed = [0.5, -0.5]

if self.rect.left <= 0 or self.rect.right >= WIDTH:
    self.speed[0] = -self.speed[0]
    self.rect.centerx += self.speed[0] * time
if self.rect.top <= 0 or self.rect.bottom >= HEIGHT:
    self.speed[1] = -self.speed[1]
    self.rect.centery += self.speed[1] * time
if pygame.sprite.collide_rect(self, pala_jug):
    self.speed[0] = -self.speed[0]
    self.rect.centery += self.speed[0] * time
if pygame.sprite.collide_rect(self, pala_cpu):
    self.speed[0] = -self.speed[0]
    self.rect.centery += self.speed[0] * time
return puntos

```

```

class Pala (pygame.sprite.Sprite):
    def __init__(self, x):
        pygame.sprite.Sprite.__init__(self)
        self.image = load_image("img/pala.png")
        self.rect = self.image.get_rect()
        self.rect.centerx = x
        self.rect.centery = HEIGHT / 2
        self.speed = 0.5
    def mover (self, time, keys):
        if keys[K_q]:
            sys.exit(0)
        if self.rect.top >= 0:
            if keys[K_UP]:
                self.rect.centery -= self.speed * time
        if self.rect.bottom <= HEIGHT:
            if keys[K_DOWN]:
                self.rect.centery += self.speed * time
    def ia (self, time, ball):
        if ball.speed[0] >= 0 and ball.rect.centerx >= WIDTH / 2:
            if self.rect.centery < ball.rect.centery:
                self.rect.centery += self.speed * time
            if self.rect.centery > ball.rect.centery:
                self.rect.centery -= self.speed * time

```

```

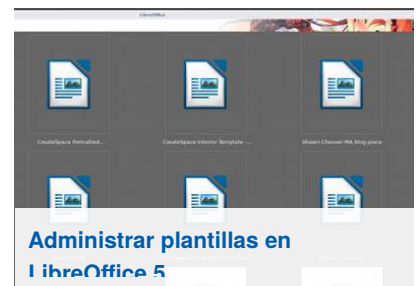
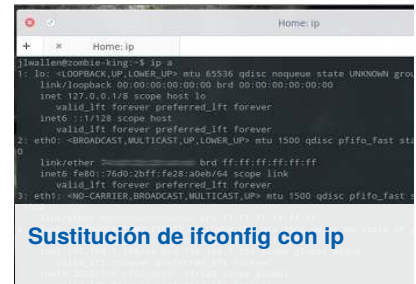
class Text (pygame.font.Font):
    def __init__(self, FontName = None, FontSize = 30):
        pygame.font.init()
        self.font = pygame.font.Font(FontName, FontSize)
        self.size = FontSize

```

```

def render (self, surface, text, color, pos):
    text = unicode(text, "UTF-8")
    x, y = pos
    for i in text.split("r"):
        surface.blit(self.font.render(i, 1, color), (x, y))
        y += self.size

```



```

# -----
# -----

# Funciones
def load_image(filename, transparent=False):
    try: image = pygame.image.load(filename)
    except pygame.error, message:
        raise SystemExit, message
    image = image.convert()
    if transparent:
        color = image.get_at((0,0))
        image.set_colorkey(color, RLEACCEL)
    return image
# -----

# -----

def main():
    screen = pygame.display.set_mode((WIDTH, HEIGHT))
    pygame.display.set_caption("Pruebas Pygame")
    background_imagen = load_image('img/fondo_pong.png')
    bola = Bola()
    clock = pygame.time.Clock()
    pala_jug = Pala(30)
    pala_cpu = Pala(610)
    text = Text()
    puntos = [0, 0]


    while True:
        time = clock.tick(60)
        keys = pygame.key.get_pressed()
        for eventos in pygame.event.get():
            if eventos.type == quit:
                sys.exit(0)
        bola.actualizar(time, pala_jug, pala_cpu, puntos)
        pala_jug.mover(time, keys)
        pala_cpu.ia(time, bola)
        screen.blit(background_imagen, (0, 0))
        numero1=str(puntos[1])
        text.render(screen, "P1: "+numero1, white, (20, 0))
        numero0=str(puntos[0])
        text.render(screen, "Cpu: "+numero0, white, (560, 0))
        screen.blit(bola.image, bola.rect)
        screen.blit(pala_jug.image, pala_jug.rect)
        screen.blit(pala_cpu.image, pala_cpu.rect)
        pygame.display.flip()
    return 0

if __name__ == '__main__':
    main()

```

 [Compartir](#)

 [Twittear](#)

 Fuentes de Información - Como programar un juego paso a paso con Pygame

 Tutorial de iniciacion [Losersjuegos]

