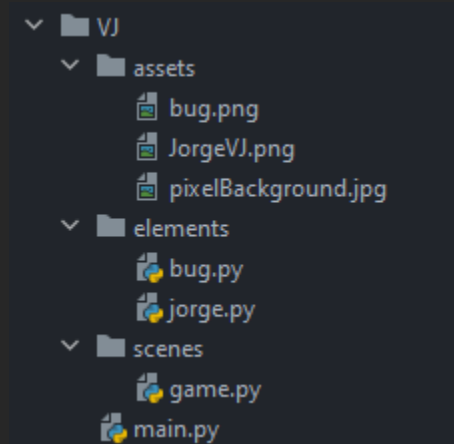


Experiencia: VideoJuegos

1.- El proyecto

Tienen una carpeta llamada VJ, con su editor de código deben abrirla como proyecto, una vez todo abierto en la vista de su proyecto (laterales) deberían ver algo similar a esto.



Esta distribución de módulos y carpetas se puede ver aterrador, pero no se preocupen! Es solo una buena práctica y nos permitirá visualizar mejor los elementos que conforman nuestro juego.

Ahora a explicar un poco que es todo lo que le hemos entregado...

1.1- Y que es todo esto???

Todos los módulos tienen un pequeño comentario explicando su función en el proyecto.

```
"""  
    Hola este es modulo game,  
    este modulo manejara la escena donde ocurre nuestro juego  
    """  
  
*modulo scenes/game.py
```

Ciertos módulos importan Pygame y Random, también ciertas teclas las que importamos directamente para más fácil acceso.

```
. import pygame  
import random  
  
from pygame.locals import (  
    K_UP, K_DOWN, K_LEFT, K_RIGHT, K_ESCAPE, KEYDOWN, QUIT, RLEACCEL)
```

Las variables para utilizar ya están presentes, las que son de tipo `None`, ustedes van a tener que definir las correctamente.

```
BUGpng = None  
BUGpng_scaled = None
```

Para la mecánica del juego van a tener que modelar dos clases y uno de sus métodos, init ya se lo entregamos.

Update va a actualizar los elementos del juego cada frame, su posición, velocidad

```
class Enemy(pygame.sprite.Sprite):
    def __init__(self, SCREEN_WIDTH, SCREEN_HEIGHT):
        # nos permite invocar métodos o atributos de Sprite
        super(Enemy, self).__init__()
        self.surf = BUGpng_scaled
        self.surf.set_colorkey((0, 0, 0), RLEACCEL)
        # la posición inicial es generada aleatoriamente, al igual que la velocidad
        self.rect = self.surf.get_rect(
            center=(
                SCREEN_WIDTH + 100,
                random.randint(0, SCREEN_HEIGHT),
            )
        )
        self.speed = random.randint(3, 5)

    def update(self):
        pass
```

- [bug.py](#)

```
JorgePNG = pygame.image.load('../assets/JorgeVJ.png')
JorgePNG_scaled = pygame.transform.scale(JorgePNG, (80, 80))

class Player(pygame.sprite.Sprite):
    def __init__(self, SCREEN_WIDTH, SCREEN_HEIGHT):
        # nos permite invocar métodos o atributos de Sprite
        super(Player, self).__init__()
        self.surf = JorgePNG_scaled
        self.surf.set_colorkey((0, 0, 0), RLEACCEL)
        self.rect = self.surf.get_rect()
        self.screen_width = SCREEN_WIDTH
        self.screen_height = SCREEN_HEIGHT

    def update(self, pressed_keys):
        pass
```

- [jorge.py](#)

Para hacer un proyecto con Pygame hay que iniciar sus módulos, esto lo hacemos con pygame.init()

```
''' iniciamos los módulos de pygame'''

pygame.init()
```

- [game.py](#)

Qué es un juego sin una pantalla? Nada.

Como estamos jugando con interfaces gráficas, vamos a indicarle al programa las características de nuestra pantalla que genere un display y que imagen va a mostrar la pantalla.

```
pygame.init()

''' Creamos y editamos la ventana de pygame (escena) '''
''' 1.-definir el tamaño de la ventana'''
SCREEN_WIDTH = 1000
SCREEN_HEIGHT = 700

''' 2.- crear el objeto pantalla'''
screen = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT))
background_image = pygame.image.load("assets/pixelBackground.jpg").convert()
```

Todo el siguiente código está en **game.py**

- [game.py](#)

Ustedes deberán desarrollar, **el Gameloop**,

Como mencionamos en la presentación, los juegos cuentan con un reloj interno, clock va a ser nuestro reloj.

```
''' Preparamos el gameloop '''
''' 1.- creamos el reloj del juego'''

clock = None
```

Como vamos hacer que aparezcan enemigos??? Bueno vamos a crear un evento ADDENEMY que se encargara de eso.

```
''' 2.- generador de enemigos'''  
  
ADDENEMY = None
```

No nos debemos olvidar de crear la instancia de nuestro jugador, también por conveniencia vamos a crear contenedores de sprites, enemies y all_sprites que nos permitirán actualizarlos mas rápidamente.

```
''' 3.- creamos la instancia de jugador'''  
player = None  
  
''' 4.- contenedores de enemigos y jugador'''  
enemies = None  
all_sprites = None
```

¡Una vez que tengamos listo todo lo anterior y completemos el gameloop vamos a tener listo nuestro primer videojuego!

```
''' hora de hacer el gameloop '''
```

2.0- El código

Ahora que entienden el código que les entregamos podemos comenzar a codificar el juego.

Podemos asegurarnos que Pygame esta bien instalado ejecutando el siguiente código por la CMD(console de comandos)

```
python -m pygame.examples.aliens
```

2.1 – El display parte 1 (READY)

Vamos a crear la pantalla donde va a ocurrir el juego, vamos a llamar a display que pertenece a Pygame, y que tiene el método set_mode que recibe una tupla con el tamaño de la pantalla.

```
pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT))
```

Para eso le asignaremos valores a las variables SCREEN_WIDTH y SCREEN_HEIGHT según las medidas de nuestro fondo 1000x700.

```
''' 1.-definir el tamaño de la ventana'''  
SCREEN_WIDTH = 1000  
SCREEN_HEIGHT = 700
```

En esta sección
usaremos el
módulo **game.py**

Y definiremos nuestra screen, usando el código que vimos más arriba.

```
''' 2.- crear el objeto pantalla'''  
screen = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT))
```

Si corres el código ahora, se abrirá una pantalla en negro durante unos instantes.

2.2.- El gameplay (IMPORTANTE)

Y que juego hacemos solo con la pantalla?

Ninguno 😊

Es por eso que debemos comenzar a crear lo que le dé forma a nuestro juego, nuestro Gameloop y todos sus elementos, como mostramos en la presentación, vamos a tener a nuestro personaje (Jorge) moviéndose por toda la ventana, esquivando obstáculos (Bugs) que aparecerán de manera aleatoria a los costados de nuestra ventana. Si un Bug le alcanza a dar a Jorge el juego termina.

Y bueno como traducimos esto al código?

Vamos a tener dos clases, la de **Player** y la de **Enemy** que.

-**Player** va a poder moverse mientras este dentro de la ventana.

-**Enemy** aparecerá cada cierto tiempo en el borde de nuestra ventana.

-Si **Player** impacta **Enemy**, se cerrará la venta

2.3.- Game loop parte 1

Para poder realizar el gameplay de nuestro juego vamos a tener que generar un loop, un game loop, que nos permitirá registrar todos los eventos que pasen. ¿Qué eventos tenemos ya? Los de

registro de input de usuario, específicamente usaremos `KEYDOWN` que registra cuando una tecla esta presionada y `QUIT` que registra cuando apretamos el botón de cierre de ventana.

Todos los eventos que se registren entre las iteraciones (repeticiones) del game loop, quedan registradas en `pygame.event.get():`, esto nos permitirá iterar sobre cada evento que haya ocurrido.

Entonces nuestro game loop debería verse como el siguiente ejemplo:

```
''' hora de hacer el gameloop '''
running = True

while running:

    for event in pygame.event.get():
        pass
```

Los eventos guardan información que se puede acceder, como por ejemplo que tipo de evento ocurrió. Con esto podemos saber que tecla fue apretada por ejemplo, si actualizamos el loop anterior para que registre los eventos que mencionamos, obtenemos algo como.

```
# variable booleana para manejar el loop
running = True

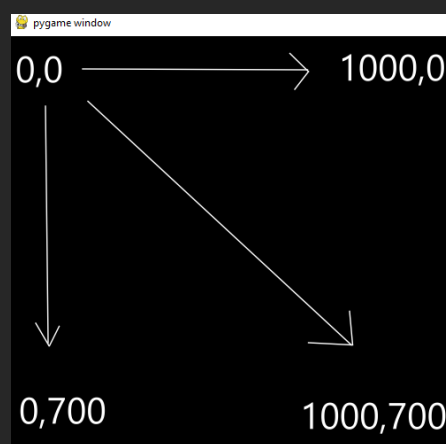
# loop principal del juego
while running:
    # iteramos sobre cada evento en la cola
    for event in pygame.event.get():
        # se presiona una tecla?
        if event.type == KEYDOWN:
            # era la tecla de escape? -> entonces terminamos
            if event.key == K_ESCAPE:
                running = False

        # fue un click al cierre de la ventana? -> entonces terminamos
        elif event.type == QUIT:
            running = False
```


Ahora nuestra ventana seguirá existiendo hasta que apretemos el ESCAPE o cerremos la ventana.

2.4 – El display parte 2

Recuerdan que los programadores comienzan contando desde el 0? Pues los display también. La ventana que creamos tiene las siguientes posiciones, como una matriz, que nos permitirá poner objetos sobre ella dando las coordenadas correspondientes.



Bueno ahora que masterizamos nuestra poderosa ventana en negro, es hora de darle un toque más colorido, vamos a agregar una imagen de fondo.

Cual es un buen lugar para que Jorge esquive los BUGS? Pues dentro de un computador! Para el fondo del juego van a usar la imagen que les dimos  pixelBackground.jpg .

Y como es que cargamos una imagen en Pygame? Pues...

```
screen = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT))
background_image = pygame.image.load("assets/pixelBackground.jpg").convert()
```

- Scenes/game.py

Que es todo ese string dentro del método `load()` ? pues es la ruta de la imagen...

el camino que debe seguir el código para encontrar el archivo, ingresamos a la carpeta `/assets` y encontramos la foto que queremos.

Que hace `.convert()` ? Escalara la imagen para que sea proporcional con el tamaño de la ventana... En este caso no hará nada ya que la ventana que creamos es del mismo tamaño que la imagen ya que escalando se puede perder calidad en la imagen.

Pero esto no es todo, debemos ahora “dibujar” la imagen en la ventana, para hacerlo dentro del loop escribiremos:

```
while running:
    screen.blit(background_image, [0, 0])
```

Que hace `screen.blit()` ? Toma dos argumentos, el QUE se dibujara y el DONDE en la superficie que escojamos, en este caso screen.

Ahora si ejecutamos no habrá pasado nada, porque?

2.4 – El display parte 2 (TODO)

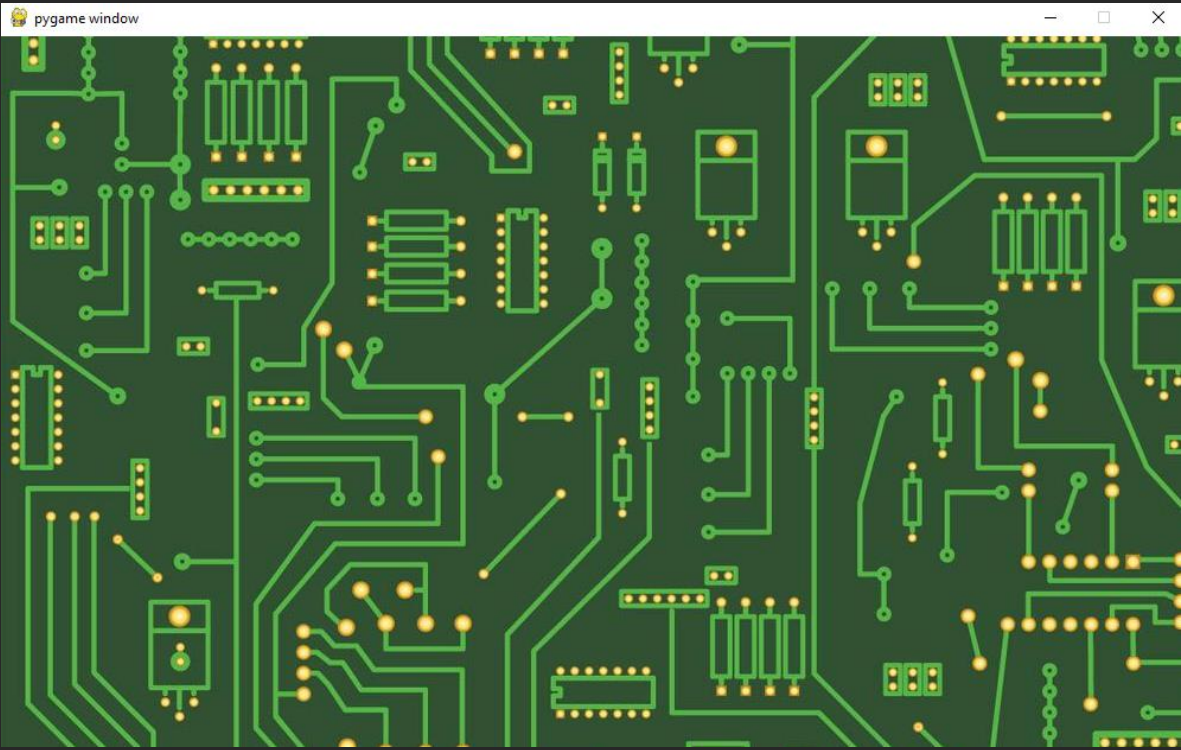
Pues le pasamos la información a la ventana pero no la volvimos a dibujar, hay que actualizarla en el loop para que la pantalla refleje todos los cambios que ocurrieron durante el Game loop.

Entonces al final del game loop hay que escribir.

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

Que hace `.flip()` ? Actualiza la ventana con todo lo que le hemos dibujado usando `blit()`.

Y listo ya hemos puesto nuestra primera ventana con fondo!



2.5 – Jorge y los BUGS (READY)

Nuevamente tenemos el problema, es entretenido como juego lo que llevamos? Aun no, hay que añadirle a Jorge y los BUGS.

```
class Enemy(pygame.sprite.Sprite):
    def __init__(self, SCREEN_WIDTH, SCREEN_HEIGHT):
        super(Enemy, self).__init__()
        pass

    def update(self):
        pass
```

```
class Player(pygame.sprite.Sprite):
    def __init__(self, SCREEN_WIDTH, SCREEN_HEIGHT):
        # nos permite invocar métodos o atributos de Sprite
        super(Player, self).__init__()
        pass

    def update(self, pressed_keys):
        pass
```

- bug.py

- jorge.py

“Ambas clases tienen algo raro no? Pues es que heredan, ¿Que es heredar? Aun no lo nescesitais”

-Jorge (posiblemente)

Lo que les interesa y les sirve... Va a permitir relacionar nuestro código con un Sprite. Pero hay que definir que vamos a usar como Sprite, en su código encontraran cuatro variables.

```
BUGpng = pygame.image.load('../assets/bug.png')
BUGpng_scaled = pygame.transform.scale(BUGpng, (64, 64))
```

```
JorgePNG = pygame.image.load('../assets/JorgeVJ.png')
JorgePNG_scaled = pygame.transform.scale(JorgePNG, (80, 80))
```

- bug.py

- jorge.py

Al igual como lo hicimos con el fondo, se usa `pygame.image.load()`. Con las variables BUGpng y JorgePNG.

Para que son las scaled? Si queremos hacer los enemigos o a Jorge más grandes, más pequeños podremos escalar la imagen que tenemos.

```
BUGpng = pygame.image.load('assets/bug.png')
BUGpng_scaled = pygame.transform.scale(BUGpng, (64, 64))

JorgePNG = pygame.image.load('assets/JorgeVJ.png')
JorgePNG_scaled = pygame.transform.scale(JorgePNG, (80, 80))
```

Que hace `pygame.transform.scale()`? Toma dos parámetros la imagen de referencia y su nuevo tamaño y entrega la imagen corregida.

Ahora podemos empezar a trabajar con las clases, empezando con `Player`.

```
def __init__(self, SCREEN_WIDTH, SCREEN_HEIGHT):
    # nos permite invocar métodos o atributos de Sprite
    super(Player, self).__init__()
    self.surf = JorgePNG_scaled
    self.surf.set_colorkey((0, 0, 0), RLEACCEL)
    self.rect = self.surf.get_rect()
    self.screen_width = SCREEN_WIDTH
    self.screen_height = SCREEN_HEIGHT
```

- jorge.py

Que hace todo eso que pusiste? Pues ...

Pygame utiliza “Surfaces” para dibujar, Piénselo como una hoja vacía en la que podemos poner cualquier color usando RGB (Red Green Blue) o poner sprites e imágenes.

Ponemos el Sprite en el `surf` de la clase, la siguiente línea nos permite que la imagen mantenga sus colores y `rect` nos dejara dibujarlo en la pantalla.

Y ahora vemos el `Enemy`

```
def __init__(self, SCREEN_WIDTH, SCREEN_HEIGHT):
    super(Enemy, self).__init__()
    self.surf = BUGpng_scaled
    self.surf.set_colorkey((0, 0, 0), RLEACCEL)
```

- bug.py

Y el `rect`?? Pues...

```
# la posicion inicial es generada aleatoriamente, al igual que la velocidad
self.rect = self.surf.get_rect(
    center=(
        SCREEN_WIDTH + 100,
        random.randint(0, SCREEN_HEIGHT),
    )
)
self.speed = random.randint(3, 5)
```

- bug.py

Lo que estamos haciendo es darle una posición inicial y una velocidad aleatoria, partirán 100 pixeles a la derecha de la pantalla (no podremos verlos al principio) , a una altura que desconocemos y a una velocidad de 3 a 5 pixeles por frame.

2.5 – Jorge y los BUGS (TODO)

Y bueno también queremos que se muevan, no? Para eso actualizaremos el método `update()` .

En Player, vamos a entregarle `pressed_keys` que tiene va a tener la información de las teclas presionadas, vamos a usar las flechas del teclado (se importaron!) usando `move_ip` moveremos el personaje en los ejes X e Y, también revisaremos que no se pueda salir de la pantalla.

```
def update(self, pressed_keys):
    if pressed_keys[K_UP]:
        self.rect.move_ip(0, -4)
    if pressed_keys[K_DOWN]:
        self.rect.move_ip(0, 4)
    if pressed_keys[K_LEFT]:
        self.rect.move_ip(-4, 0)
    if pressed_keys[K_RIGHT]:
        self.rect.move_ip(4, 0)

    if self.rect.left < 0:
        self.rect.left = 0
    if self.rect.right > self.screen_width:
        self.rect.right = self.screen_width
    if self.rect.top < 0:
        self.rect.top = 0
    if self.rect.bottom > self.screen_height:
        self.rect.bottom = self.screen_height
```

• jorge.py

Ahora revisaremos el `update` de `Enemy`.

Vamos a usar la `speed` que definimos antes, moveremos los BUGS a través de la pantalla, y si salen de ella se destruirán.

```
def update(self):
    self.rect.move_ip(-self.speed, 0)
    if self.rect.right < 0:
        self.kill()
```

- `bug.py`

Y si corremos el código.... NADA PASA?????

2.6 – Jorge y los Bugs parte 1.5 (TODO)

Bueno `rect` nos dejara dibujar todo lo que creamos en pantalla ... pero el no va a hacerlo, hay que usar `blit` al igual que con el fondo....

Hay que usar `blit` con cada enemigo???

Que pasa si hacemos 100???

Crearemos una instancia de `player`:

```
''' 3.- creamos la instancia de jugador'''
player = Player(SCREEN_WIDTH, SCREEN_HEIGHT)
```

- `game.py`

Pygame tiene la solución, usando `sprite.Group()` , vamos a agrupar todos nuestros objetos para que sea más fácil dibujarlos todos en la pantalla.

Aprovecharemos de crear dos, uno para todos los objetos y otro para solo los enemigos.

Porque uno solo para enemigos? Pues la usaremos más adelante tengan fe.

```
''' 4.- contenedores de enemigos y jugador'''
enemies = pygame.sprite.Group()
all_sprites = pygame.sprite.Group()
all_sprites.add(player)
```

- `game.py`

Y agregamos al game loop, antes del `pygame.display.flip()`, los siguiente:

```
# dibujamos todos los sprites
for entity in all_sprites:
    screen.blit(entity.surf, entity.rect)
```

- `game.py`

Y si corremos el código...aparece Jorge y

De nuevo no pasa nada?????

Hay que añadir los `update` al game loop.

```
# obtenemos todas las teclas presionadas actualmente
pressed_keys = pygame.key.get_pressed()

# actualizamos el sprite del jugador basado en las teclas presionadas
player.update(pressed_keys)

# actualizamos los enemigos
enemies.update()
```

- game.py

Listo ahora podemos ver y mover a Jorge!!!

2.7 – Generando BUGS (TODO)

Habrán notado que no han aparecido los bugs, es porque no hemos creado ninguna instancia de ellos así que es hora de crear un evento que nos genere bugs.

Para hacer un evento propio hay que saber que internamente Pygame tiene ordenado los eventos como números (INT), Tiene un máximo de 32 eventos pero Pygame ya usa 23, tenemos del 24 al 32 para usar como queramos, `pygame.USEREVENT` es el 24 y el que vamos a usar es el 25.

```
''' 2.- generador de enemigos'''
ADDENEMY = pygame.USEREVENT + 1
pygame.time.set_timer(ADDENEMY, 600)
```

- game.py

```
pygame.time.set_timer(ADDENEMY, 600)
```

Que es lo que hace `pygame.time.set_timer(ADDENEMY, 600)`? Colocara el evento ADDENEMY a la cola de eventos cada 600 ticks.

Pero ahora hay que crear las instancias de BUG en el game loop.

```
for event in pygame.event.get():
    # se presiona una tecla?
    if event.type == KEYDOWN:
        # era la tecla de escape? -> entonces terminamos
        if event.key == K_ESCAPE:
            running = False

        # fue un click al cierre de la ventana? -> entonces te
    elif event.type == QUIT:
        running = False

    # es un evento que agrega enemigos?
    elif event.type == ADDENEMY:
        new_enemy = Enemy(SCREEN_WIDTH, SCREEN_HEIGHT)
        enemies.add(new_enemy)
        all_sprites.add(new_enemy)
```

- game.py

Detectamos el tipo de evento, añadimos la nueva instancia de Enemy a los groups y listo!

2.8 – El tiempo

Todo pasaba muy rápido, no? Es que los ticks van a lo máximo que pueden, mientras mejor procesador más rápido el juego, Para regularlo usaremos Clock del modulo time de Pygame.

```
''' 1.- creamos el reloj del juego'''  
clock = pygame.time.Clock()
```

- game.py

Y en el game loop vamos a indicar a cuantos frames por segundo se va actualizar la ventana y los elementos del juego, añadiremos el código al final del game loop.

```
clock.tick(40)
```

Si ahora corremos el juego ira a la velocidad que le pusimos!!!

2.9 – Esquivando BUGS (TODO)

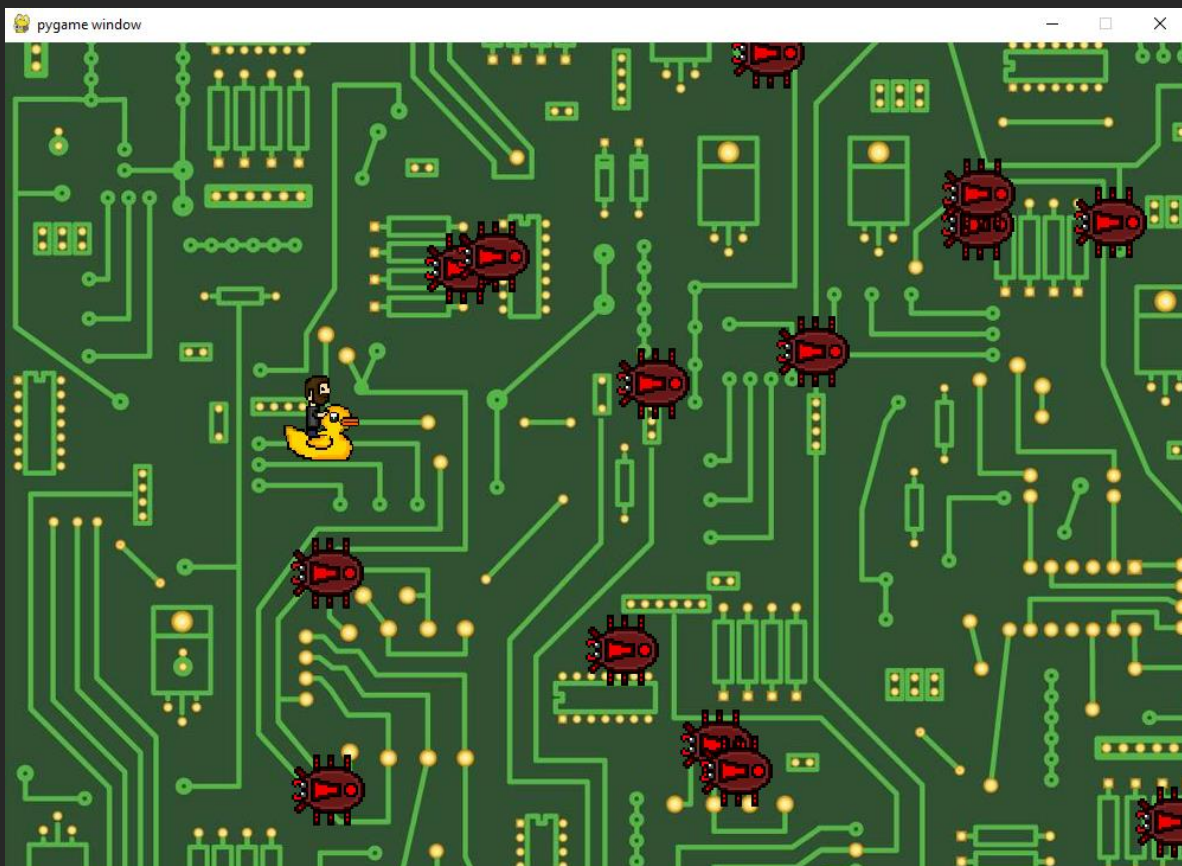
Vamos a tener que calcular el recuadro de jugador y lo de los bugs y ver si se tocan todo el rato?

No

La magia de Pygame es que vamos a usar una función que nos entrega para ver si Jorge ha chocado con los BUGS.

```
# vemos si algun enemigo a chocado con el jugador  
if pygame.sprite.spritecollideany(player, enemies):  
    # si pasa, removemos al jugador y detenemos el loop del juego  
    player.kill()  
    running = False
```

3.0 – JUGAR Y MEJORAR



**AHORA ya has programado tu primer juego! Aun tienes tiempo?
Trata de mejorarlo y añadirle funciones para personalizarlo!!!**

