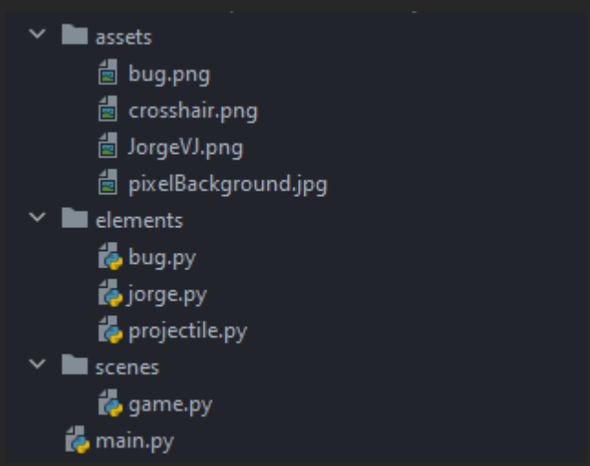
Experiencia: Videojuegos 2🎮💀

0.0 – [TIPS/ERRORES COMUNES]

- Usar la versión de Python en la que instalaron Pygame
- Estar en la raiz del proyecto ./VJ-2 para que no tengan problemas al ejecutar
- Ctrl + S (Guardar) antes de ejecutar. VSC compila la última versión guardada
- Linux/Mac: python3 main.py y pip3 install pygame (con el 3)

1.- El proyecto 2.0 💀

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



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

1.1- Ya todo listo??? Pues no ....

Comparado con **VJ-1**, obvio pero esto es **VJ-2**.

Van a iterar sobre una version terminada del juego y vamos a ir agregandole **features**

```python
class Projectile(pygame.sprite.Sprite):
    def __init__(self, pos, direction, SCREEN_WIDTH, SCREEN_HEIGHT):
        super(Projectile, self).__init__()
        pass

    def update(self):
        pass
```

**modulo elements/projectile.py*

Como que Jorge dispare para defenderse de los bugs!!!

2.0- El código

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

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

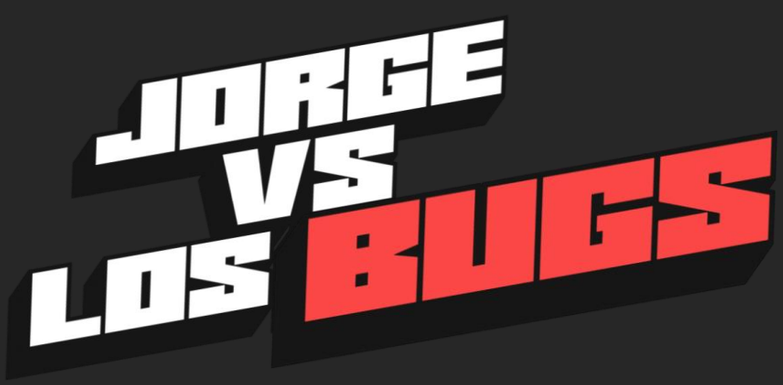
2.1 – El Juego parte 1 (READY)

En esta sección usaremos el módulo game.py

**Para más detalle leer manual VJ-1*

Jorge puede moverse con las flechas de tu teclado, aparecen los bugs y si los tocas el juego termina ....

Muy simple verdad?
ESTO ES ...



2.2 – Jorge VS BUGS (Por Hacer)

Iniciaremos creando las balas que usara Jorge para defenderse.

Esta vez nos dio lata hacer un Sprite así que usaremos . . .

"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.)*

```
class Projectile(pygame.sprite.Sprite):
    def __init__(self, pos, direction, SCREEN_WIDTH, SCREEN_HEIGHT):
        super(Projectile, self).__init__()
        self.surf = pygame.Surface((10, 10))
        self.surf.fill((255, 255, 255))
        self.rect = self.surf.get_rect(center=pos)
```

En Elements/projectile.py

De esta manera Projectile será un cubo de 10x10 de color blanco y usaremos el centro de la figura como origen para todas las transformaciones.

```
class Projectile(pygame.sprite.Sprite):
    def __init__(self, pos, direction, SCREEN_WIDTH, SCREEN_HEIGHT):
        super(Projectile, self).__init__()
        self.surf = pygame.Surface((10, 10))
        self.surf.fill((255, 255, 255))
        self.rect = self.surf.get_rect(center=pos)
        self.speed = 10
        self.direction = direction
        self.screen_width = SCREEN_WIDTH
        self.screen_height = SCREEN_HEIGHT
```

Y ahora que asignamos unas variables adicionales. . .

Podremos, mover las balas, darle dirección y que se destruyan al salir de la pantalla.

2.3 – Piew Piew Piew (POR HACER)

Ahora habrá que manejar la creación de las múltiples instancias de las balas

Pygame tiene ordenado los eventos como números (INT), Tiene un máximo de 32 eventos pero Pygame ya usa 23, usaremos el que detecta el click del mouse para generar las balas.

```
from elements.projectile import Projectile
```

Para esto se darán cuenta Que Jorge.py importa el modulo Projectile. Jorge se encargará de spawnear las balas.

En el método init agregaremos una variable para guardar y gestionar los projectiles creados.

```
self.projectiles = pygame.sprite.Group()
```

Y añadiremos un método nuevo a Jorge. Se encargará de registrar información del **Mouse,** crear una nueva instancia de projectile y agregarlo a la variable que creamos antes.

```
def shoot(self, mouse_pos):
    #* Calcula la dirección del proyectil
    direction = (mouse_pos[0] - self.rect.centerx, mouse_pos[1] - self.rect.centery)
    length = math.hypot(*direction)
    direction = (direction[0] / length, direction[1] / length)
    #* Crea un proyectil en la posición del jugador
    projectile = Projectile(self.rect.center, direction, self.screen_width, self.screen_height)
    self.projectiles.add(projectile)
```

Y en el ciclo **for** de los **eventos** agregaremos la ejecución del método.

```python
#* Dispara un proyectil si el usuario hace click
elif event.type == pygame.MOUSEBUTTONDOWN:
    player.shoot(pygame.mouse.get_pos())
```

En scenes/game.py

2.4 – Que se muevan las balas (POR HACER)

Las balas no se moverán ni se verán aun RECORDAR no hemos ejecutado update ni blit para los projectiles.

```python
while running:

    screen.blit(background_image, [0, 0])

    for entity in all_sprites:
            screen.blit(entity.surf, entity.rect)

    for projectile in player.projectiles:
        screen.blit(projectile.surf, projectile.rect)
```

En scenes/game.py

Pero aun no programamos el Update de los proyectiles.

```python
def update(self):
    #* Mueve el proyectil en la dirección dada
    self.rect.move_ip(self.direction[0] * self.speed, self.direction[1] * self.speed)
    #* Elimina el proyectil si sale de la pantalla
    if self.rect.right < 0 or self.rect.left > self.screen_width or self.rect.bottom < 0 or self.rect.top > self.screen_height:
        self.kill()
```

En elements/projectile.py

Y hacemos que cada vez que se update Jorge, también lo hagan sus proyectiles ...

```python
#* Actualiza la posición de los proyectiles
self.projectiles.update()
```

En elements/jorge.py

Y obviamente las balas han de destruir los BUGS

```python
    #* Si un proyectil golpea a un enemigo, el proyectil y el enemigo mueren
    pygame.sprite.groupcollide(player.projectiles, enemies, True, True)
```

En scenes/game.py

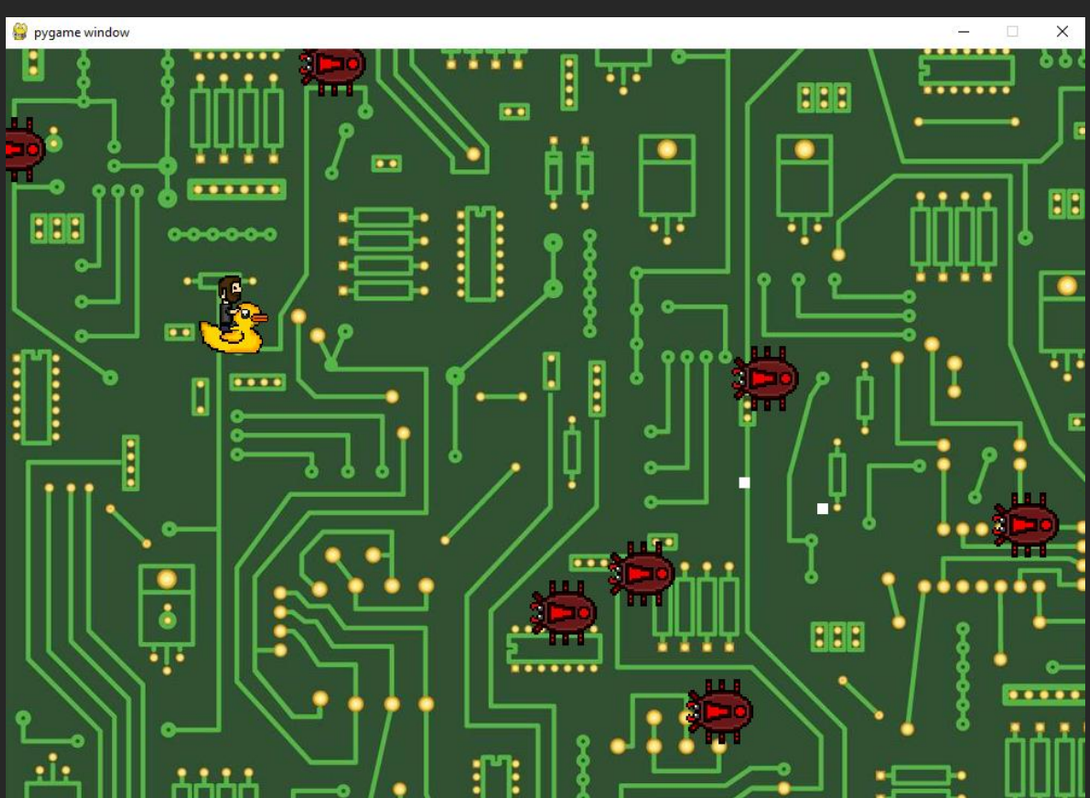
**Dentro del Gameloop! While running*

2.5 – Autonomia (POR HACER)

Ahora es hora del vuelo del pájaro...

Haz de manera autónoma una de las siguientes features

- Agregar un cursor
- Una pantalla de muerte



AHORA ya has programado tu primer juego! Aun tienes tiempo?
APUESTA POR EL HONORS!!!