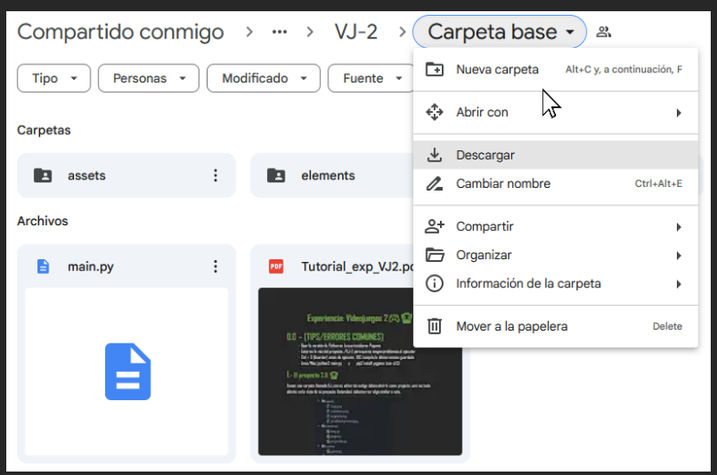
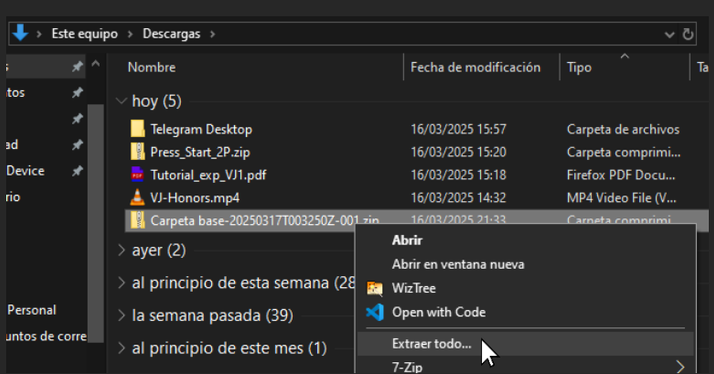
# Experiencia: Videojuegos 2

## 0.- Abrir proyecto en VSCode

Descargar la carpeta completa como aparece en al imagen de abajo.

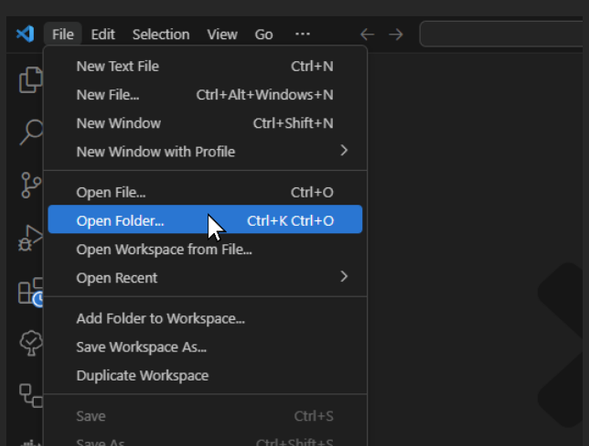



Eso les descargará un archivo zip que deben descomprimir. Si no sabes como descomprimir un archivo revisa las guias de instalacion de python que subimos.

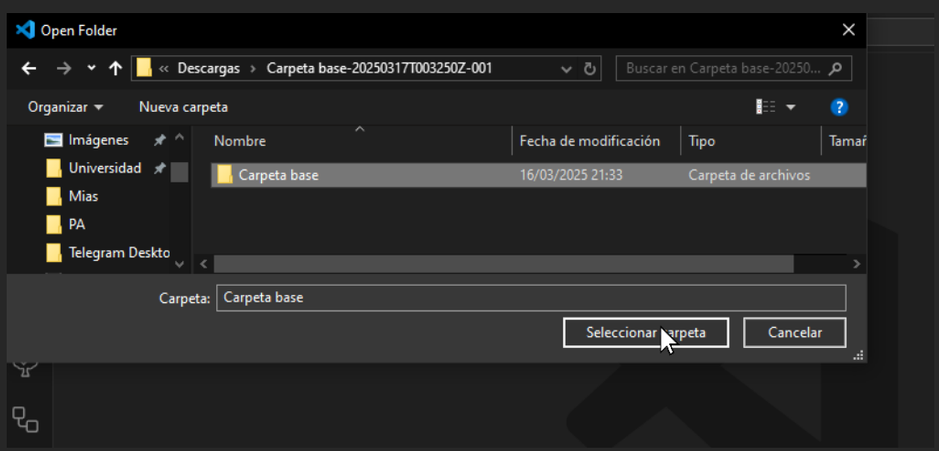



*Ejemplo descomprimir Windows (Mac deberia funcionar con doble click)*

Al descomprimir se generará una carpeta con el mismo nombre. Deben abrir Vscode y darle a "Abrir carpeta".

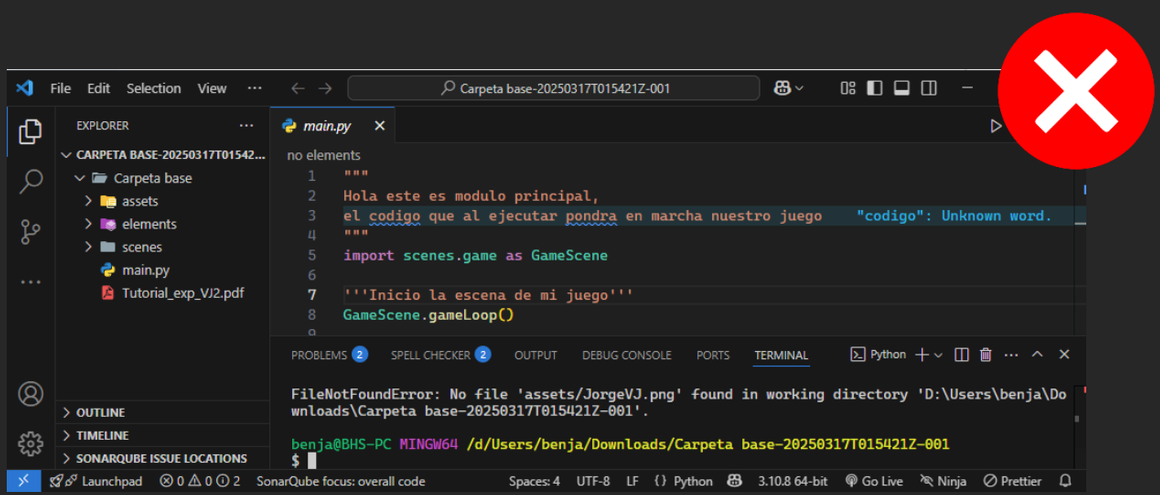



SUPER IMPORTANTE, debes abrir la carpeta que se llama exactamente "Carpeta base".

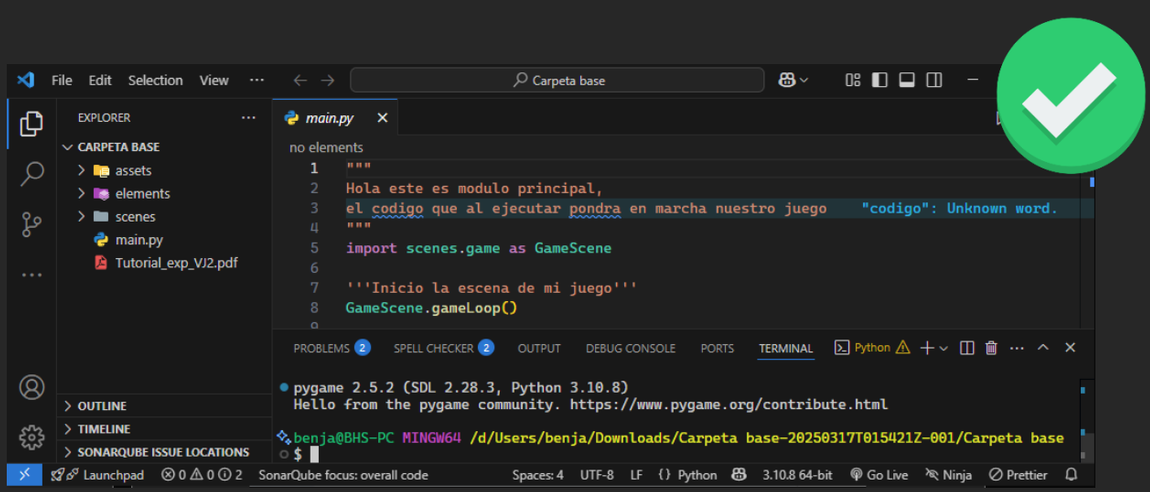



Por ejemplo acá al descomprimir generó "Carpeta base-20250317T003250Z-001" que dentro contiene otra carpeta llamada "Carpeta base". Debes entrar y seleccionar especificamente "Carpeta base".

Para verificar que abriste la carpeta correcta corre el archivo main.py




Si te aparece FileNotFoundError es que abriste la carpeta incorrecta. A la izquierda se ve que abrí "Carpeta base-20250317T003250Z-001" en vez de "Carpeta base".




Si en cambio te aparece un mensaje de pygame esta todo bien y puedes seguir con el tutorial. A la izquierda se ve que abrí directamente "Carpeta base".

En este manual encontrarás 2 tipos de secciones:

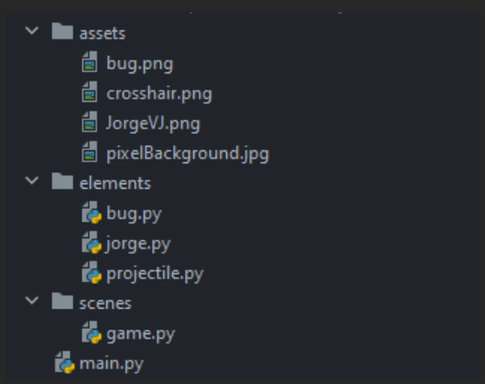
- INFO: explica funcionamiento de código ya en carpeta base
- POR HACER: código que no está en la carpeta base y debes agregar

Siempre que quieras correr el juego debes ejecutar main.py

Para entender mejor el proyecto se recomienda leer bien el manual de VJ-1

## 1.- El proyecto 2.0 (INFO)

Deberían ver algo similar a esto en su editor de codigo.

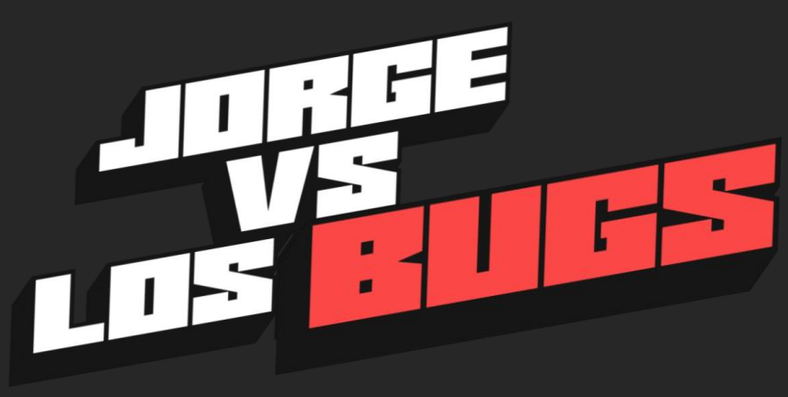



Ahora a explicar un poco que es todo lo que le hemos entregado...

## 1.1- Ya todo listo??? Pues no .... (INFO)

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

Muy simple verdad?
ESTO ES ...



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__()

        # POR HACER (2.0): Aspecto y parámetros iniciales de la bala
        pass

        # POR HACER (2.1): Parámetros iniciales de la bala
        pass


    def update(self):

        # POR HACER (2.2): Mover la bala y eliminarla si se sale de la pantalla
        pass
```

**modulo elements/projectile.py*

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

## 2.0 – Jorge VS BUGS (POR HACER)

Iniciaremos creando las balas que usará 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.)*

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

    def __init__(self, pos, direction, SCREEN_WIDTH, SCREEN_HEIGHT):
        super(Projectile, self).__init__()

        # POR HACER (2.0): Aspecto y parámetros iniciales de la bala
        self.surf = pygame.Surface((10, 10))
        self.surf.fill((255, 255, 255))
        self.rect = self.surf.get_rect(center=pos)

        # POR HACER (2.1): Parámetros iniciales de la bala
        pass
```

**En Elements/projectile.py**

De esta manera Projectile será un cubo de 10x10 de color blanco (255,255,255) y usaremos el centro de la figura como origen para todas las transformaciones.

## 2.1 – Disparando Balas (Por Hacer)

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

    def __init__(self, pos, direction, SCREEN_WIDTH, SCREEN_HEIGHT):
        super(Projectile, self).__init__()

        # POR HACER (2.0): Aspecto y parámetros iniciales de la bala
        self.surf = pygame.Surface((10, 10))
        self.surf.fill((255, 255, 255))
        self.rect = self.surf.get_rect(center=pos)

        # POR HACER (2.1): Parámetros iniciales de la bala
        self.speed = 10
        self.direction = direction
        self.screen_width = SCREEN_WIDTH
        self.screen_height = SCREEN_HEIGHT
```

En Elements/projectile.py

## Y ahora con estas variables adicionales. . .

Podremos, mover las balas, darle dirección y saber cuando se salgan de la pantalla.

## 2.2 – Actualizar las balas (POR HACER)

Ahora falta definir el método que actualizará las balas en cada tick.

```python
def update(self):

    # POR HACER (2.2): Mover la bala y eliminarla si se sale de la pantalla

    # Mueve el proyectil en la dirección indicada
    self.rect.move_ip(self.direction[0] * self.speed, self.direction[1] * self.speed)

    # Elimina el proyectil si se sale de la pantalla
    if (self.rect.left < 0                          # Se sale por la izquierda
        or self.rect.right > self.screen_width      # Se sale por la derecha
        or self.rect.top < 0                        # Se sale por arriba
        or self.rect.bottom > self.screen_height):  # Se sale por abajo
        self.kill()
```

En Elements/projectile.py

self.direction es un vector (x, y) que multiplicamos por nuestra velocidad para mover la bala.

Además queremos eliminarla si se sale de la pantalla

## 2.3 – Piew  Piew Piew (POR HACER)

Ahora habrá que manejar la creación de las múltiples instancias de las balas. Jorge se encargara de spawnear las balas. En el método init agregaremos una variable para guardar y gestionar los projectiles creados.

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

    def __init__(self, SCREEN_WIDTH, SCREEN_HEIGHT):
        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

        # POR HACER (2.3): Crear lista de proyectiles
        self.projectiles = pygame.sprite.Group()
```

En elements/jorge.py

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

```python
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

    # POR HACER (2.3): Actualizar la posición de los proyectiles
    self.projectiles.update()
```

En elements/jorge.py

Y añadiremos un método nuevo a Jorge. Se encargará de dada una posición calcular que dirección debe tener el proyectil, crear una nueva instancia de projectile y agregarlo a la lista

```python
def shoot(self, mouse_pos):

    # POR HACER (2.3): Crear y calcular dirección proyectil

    # Calcula la dirección del proyectil
    direction = (mouse_pos[0] - self.rect.centerx, mouse_pos[1] - self.rect.centery)
    length = math.hypot(direction[0], direction[1])
    direction = (direction[0] / length, direction[1] / length)

    # Crea el proyectil y lo añade al grupo de proyectiles
    projectile = Projectile(self.rect.center, direction, self.screen_width, self.screen_height)
    self.projectiles.add(projectile)
```

En elements/jorge.py

## 2.4 – Detectar el disparo (POR HACER)

Tenemos creado nuestro proyectil y Jorge puede disparar. Nos falta conectar el click al disparo de Jorge.

Pygame tiene un evento que detecta el click del mouse, lo usaremos para generar las balas. En el ciclo **for** de los **eventos** agregaremos la ejecución del método.

```python
# iteramos sobre cada evento en la cola
for event in pygame.event.get():
    # se presiono 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

    elif event.type == ADDENEMY:
        new_enemy = Enemy(SCREEN_WIDTH, SCREEN_HEIGHT)
        enemies.add(new_enemy)
        all_sprites.add(new_enemy)

    # POR HACER (2.4): Agregar evento disparo proyectil
    elif event.type == pygame.MOUSEBUTTONDOWN:
        player.shoot(pygame.mouse.get_pos())
```

En scenes/game.py

## 2.5 – Que se muevan las balas (POR HACER)

Las balas se moverán, porque el update de Jorge también actualiza las balas. Sin embargo, no se verán aun RECORDAR no hemos ejecutado blit para los projectiles.

En scenes/game.py

```
# loop principal del juego

while running:

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

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

    # POR HACER (2.5): Pintar proyectiles en pantalla
    for projectile in player.projectiles:
        screen.blit(projectile.surf, projectile.rect)
```

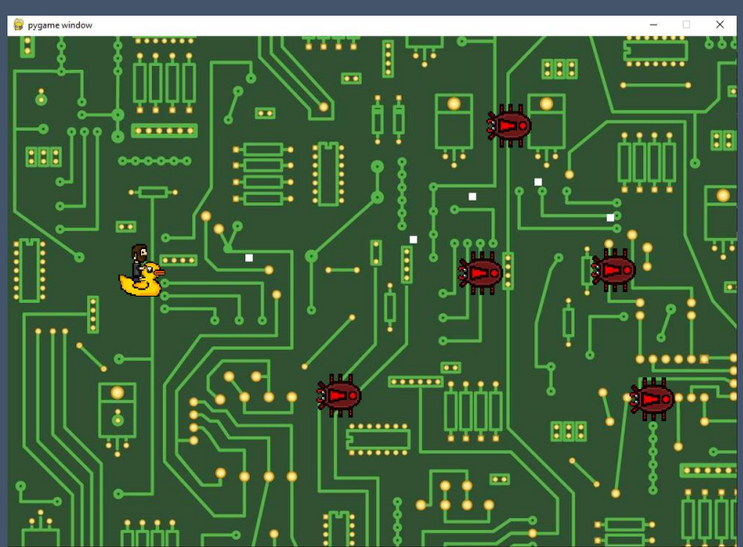
Y obviamente las balas han de destruir los BUGS

En scenes/game.py

```
while running:

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

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

    # POR HACER (2.5): Pintar proyectiles en pantalla
    for projectile in player.projectiles:
        screen.blit(projectile.surf, projectile.rect)

    # POR HACER (2.5): Eliminar bug si colisiona con proyectil
    pygame.sprite.groupcollide(enemies, player.projectiles, True, True)
```

Ahora si colisiona un bug y un proyectil ambos serán eliminados del juego

Ya funciona nuestro disparo!!!

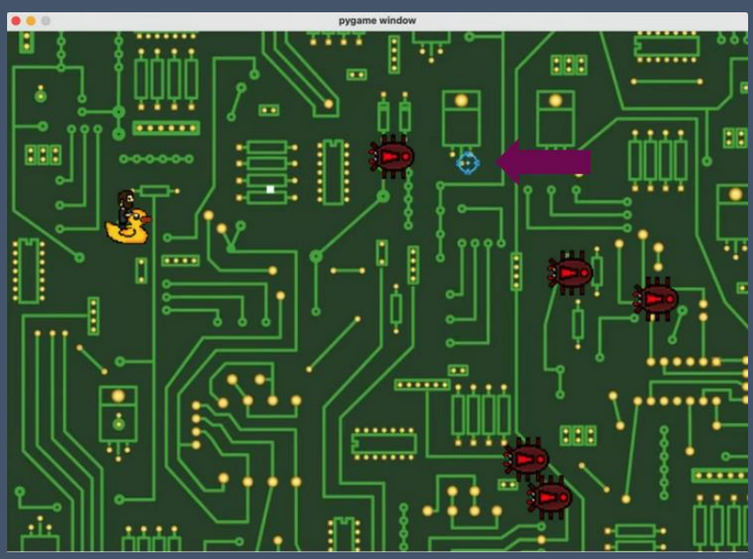


## 3.0 – Autonomia (POR HACER)

Ahora es hora del vuelo del pájaro...

Haz de manera autónoma las siguientes features

- Agregar una mirilla (algo que indique la posición del mouse)
- Crear una pantalla de muerte (DEBE ESTAR EN OTRO ARCHIVO DE SCENES Y TENER SU PROPIO GAMELOOP)
- Algún tipo de cooldown para el disparo (por ejemplo, disparar una vez y tener que esperar x tiempo)





AHORA ya has programado tu primer juego!

Aun tienes tiempo? APUESTA POR EL HONORS!!!

Piensa cómo agregarías más enemigos, un menú de inicio, nuevas habilidades, etc