

Trabajo Práctico Final — Programación Computacional

Flappy Fish: Juego basado en Pygame

Julieta Zaroni, Mariia Osipova, Santino Scofano y Morena Roldan

Universidad de San Andrés

jzanoni@udesa.edu.ar
mosipova@udesa.edu.ar
sscofano@udesa.edu.ar
mroldan@udesa.edu.ar

12 de diciembre 2025

Resumen de la presentación

1 Introducción

2 Arquitectura del Juego

3 Modo manual del juego

Módulo game.py

Módulo fish.py

Módulo generacion_de_tuberias.py

Módulo menu.py

Module main.py

Introducción

Nuestro trabajo práctico está dividido en dos partes: la primera es sobre el desarrollo de un videojuego manual inspirado en Flappy Bird, llamado Flappy Fish, implementado en Python con la librería Pygame.



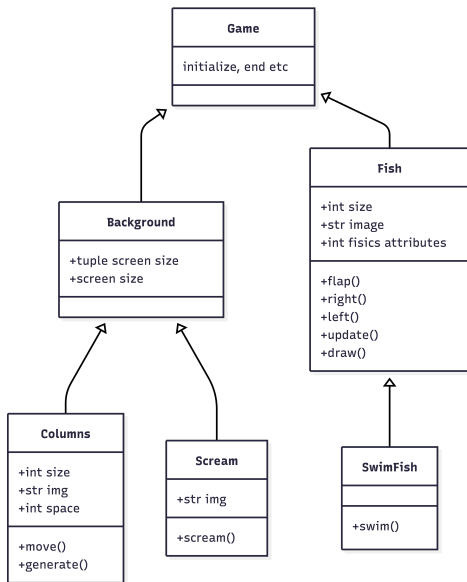
Introducción

La segunda parte se enfoca en la implementación de un Algoritmo Genético (AG) para entrenar a una población de “peces”, se juega de forma autónoma al videojuego manual.



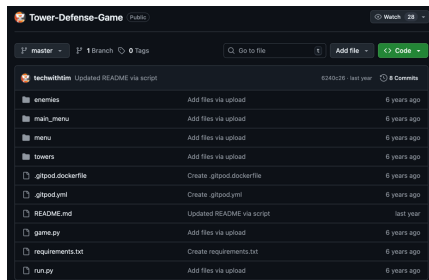
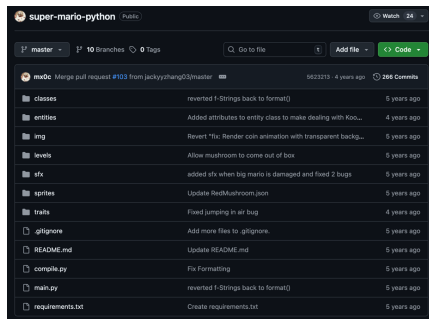
Arquitectura del Juego

Pensando en la arquitectura del juego, nos enfrentamos al primer desafío: ¿cómo debíamos estructurar y organizar el proyecto? Comenzamos trabajando a partir de este borrador inicial.

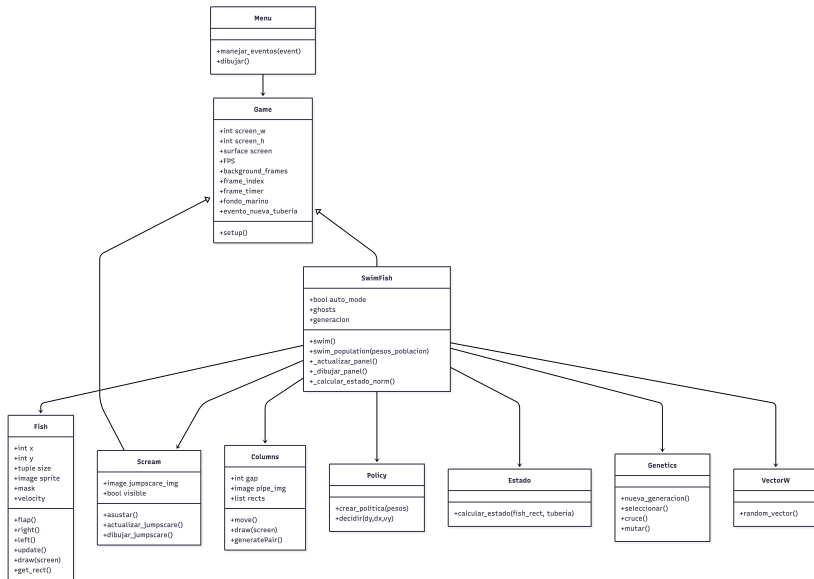


Arquitectura del Juego

Analizamos varios juegos desarrollados con Pygame y publicados de forma abierta. Estas referencias nos permitieron observar enfoques comunes de arquitectura y organización del código. Entre ellos, miramos proyectos como [Super Mario Python](#) y [Tower Defence Game](#), que utilizamos como guía conceptual.



Arquitectura del Juego



Arquitectura del Juego

El proyecto se estructura en los siguientes módulos:

- **game.py** — clase base del juego (ventana, fondo, sonido y tuberías).
- **fish.py** — física, movimiento y máscara de colisión del pez.
- **screamer.py** — módulo del screamer para el juego.
- **generacion_de_tuberias.py** — creación, posición y movimiento de las tuberías.
- **menu.py** — interfaz del menú principal y manejo de opciones.
- **swim_fish.py** — lógica del juego (manual/modo con Algoritmo Genético).
- **ml/** — política del agente, cálculo del estado, generación de pesos y genética.

Módulo game.py (I)

La clase `Game` actúa como el marco general del que hereda `SwimFish`. Su constructor realiza la configuración inicial del entorno:

- Inicializa Pygame y el objeto `Clock` (FPS = 120).
- Crea la ventana principal del juego (1000×600 píxeles).
- Carga los fotogramas del fondo animado.

```
class Game:
    def __init__(self):
        pygame.init()
        self.clock = pygame.time.Clock()
        self.FPS = 120
        self.screen = pygame.display.set_mode((1000, 600))

        # Animación de fondo
        self.animation_folder = "../data/img/fondo_animado"
        self.background_frames = self._load_background_frames()
        self.frame_index = 0
        self.frame_rate = 30
```

Módulo game.py (II)

- Define la música de fondo y sonido del salto.
- Sprite de tubería y máscara pixel-perfect.

```
self.music_path = "../data/audios/linkin park fondo.ogg"  
pygame.mixer.music.load(self.music_path)
```

```
self.sonido_salto = pygame.mixer.Sound(  
    "../data/audios/efecto bubble.ogg")
```

```
# Fondo marino  
self.fondo_marino = pygame.image.load(  
    "../data/img/pixil-frame-0.png").convert()  
self.fondo_marino = pygame.transform.scale(  
    self.fondo_marino, (self.screen_w, self.screen_h))
```

Módulo game.py (III)

- Define el hueco vertical entre tuberías = 300.
- Define el evento `evento_nueva_tuberia` cada 1500 ms.

```
# Tuberías
self.imagen_tuberia = pygame.image.load(
    "../data/img/alga2.png").convert_alpha()
self.imagen_tuberia = pygame.transform.scale(
    self.imagen_tuberia, (70, 400))
self.tuberia_mask = pygame.mask.from_surface(
    self.imagen_tuberia)
self.hueco_entre_tuberias = 300

self.evento_nueva_tuberia = pygame.USEREVENT
pygame.time.set_timer(self.evento_nueva_tuberia, 1500)
```

Entonces, Game sabe todo sobre la ventana, el fondo y las tuberías, pero no sabe nada sobre quién está volando alrededor de este mundo o cómo se desarrolla exactamente el juego; esa es responsabilidad de SwimFish.

Módulo fish.py: Fish y su fisica (I)

El Fish en fish.py una clase independiente que luego se utiliza dentro de otras clases (principalmente dentro de SwimFish) como un objeto compuesto.



- En el constructor se carga la imagen del pez, se escala al tamaño indicado y se crean su `rect` y la máscara de colisión.

```
class Fish:
    def __init__(self, x, y, size, image):

        self._start_pos = (x, y)

        self.size = size
        self.original_image = pygame.image.load(
            image).convert_alpha()
        self.original_image = pygame.transform.scale(
            self.original_image, (size[0], size[1]))
        self.image = self.original_image
        self.rect = self.image.get_rect(center = (x, y))
        self.mask = pygame.mask.from_surface(self.image)
```

Módulo fish.py: Fish y su fisica (II)

- El pez posee una velocidad vertical, una gravedad constante, una fuerza de salto, (donde un valor negativo indica movimiento ascendente) y una velocidad máxima de caída.
- El método `flap()` simplemente reinicia la velocidad al valor de `jump_strength`, produciendo un impulso instantáneo hacia arriba.

```
self.velocity = 0
self.gravity = 0.3
self.jump_strength = -10
self.max_fall_speed = 100
self.air_resistance = 0.9

def flap(self):
    self.velocity = self.jump_strength
```

Módulo fish.py: Fish y su fisica (III)

- El método `update()` incrementa la velocidad con la gravedad, la limita según `max_fall_speed`, actualiza la posición vertical del `rect` y recalcula la rotación del sprite: el ángulo es proporcional a la velocidad, pero está acotado aproximadamente entre -30° durante el ascenso y $+90^\circ$ durante la caída. Tras la rotación, se recalculan el `rect` y la máscara.

```
def update(self):  
    self.velocity += self.gravity  
  
    if self.velocity > self.max_fall_speed:  
        self.velocity = self.max_fall_speed  
  
    self.rect.y += self.velocity  
    self._rotar_pez()
```

Módulo fish.py: Fish y su fisica (IV)

- El método `_rotar_pez()` controla la orientación visual del sprite según su velocidad vertical. Calcula un ángulo proporcional a la velocidad: cuando el pez asciende se limita a -30° , y cuando cae a $+90^\circ$. Rota la imagen original con `pygame.transform.rotate()` y actualiza el centro del `rect` para mantener la posición. Finalmente, se recalcula la máscara de colisión a partir de la imagen rotada.

```
def _rotar_pez(self):
    angulo = self.velocity * 3
    if self.velocity > 0:
        angulo = min(angulo, 90)
    else:
        angulo = max(angulo, -30)
    self.image = pygame.transform.rotate(
        self.original_image, -angulo)
    old_center = self.rect.center
    self.rect = self.image.get_rect(center=old_center)
    self.mask = pygame.mask.from_surface(self.image)
```

Módulo fish.py: Fish y su fisica (V)

- El método `reset()` restablece el pez a su posición inicial, reinicia su velocidad y reconstruye el `rect` y la máscara originales. Se usa en el caso de una colision para reiniciar el juego.

```
def reset(self):  
    self.rect.center = self._start_pos  
    self.velocity = 0  
    self.image = self.original_image  
    self.rect = self.image.get_rect(center=self._start_pos)  
    self.mask = pygame.mask.from_surface(self.image)
```

Módulo generacion_de_tuberias.py (I)

La clase `tuberias` en `generacion_de_tuberias.py` define un par de obstáculos: el tubo de algas superior e inferior.



- Se selecciona una `altura_referencia` en el rango `[150, 450]`;
- A partir de esa posición se construyen los rectángulos de dos tuberías, tal que entre ambos se mantenga un hueco vertical de altura `hueco` (parámetro recibido desde `Game`).

```
class tuberias:
    def __init__(self, x, hueco, imagen):
        self.imagen_tuberia = imagen
        self.altura_referencia = random.randint(150, 450)
        self.x = x
        self.hueco = hueco
        self.tubo_arriba = self.imagen_tuberia.get_rect(
            midbottom = (
                x, self.altura_referencia - hueco // 2))
        self.tubo_abajo = self.imagen_tuberia.get_rect(
            midtop = (
                x, self.altura_referencia + hueco // 2))
```

Módulo generacion_de_tuberias.py (II)

- El método `mover_tuberias()` simplemente desplaza la coordenada `x` y sincroniza ese movimiento con los rectángulos de ambas tuberías.
- El método `dibujar_tuberias()` dibuja la tubería superior invertida y la inferior en su orientación normal.

```
def mover_tuberias(self):  
    self.x -= self.velocidad  
    self.tubo_arriba.x = self.x  
    self.tubo_abajo.x = self.x  
  
def dibujar_tuberias(self, screen):  
    screen.blit(pygame.transform.flip(self.imagen_tuberia, False,  
    screen.blit(self.imagen_tuberia, self.tubo_abajo)
```

Módulo generacion_de_tuberias.py (III)

- Ambas tuberías comparten una misma coordenada horizontal x , que posteriormente se reduce a una velocidad constante `velocidad = 4`, pidiendo que las tuberías se desplacen de derecha a izquierda.
- Se almacena `gap_y = altura_referencia`, es decir, la coordenada vertical del centro del pasaje, utilizada para calcular la característica `dy` del agente.

```
self.velocidad = 4  
self.gap_y = self.altura_referencia
```

Módulo menu.py (I)

El menú en `menu.py` recibe la pantalla y sus dimensiones, y crea el título `FLAPPY FISH!` junto con dos opciones: o el juego manual o la simulación (AG).



Módulo menu.py (II)

El método `manejar_eventos()` devuelve el modo de juego cuando el usuario hace clic con el mouse sobre la opción correspondiente o al presionar las teclas 1/2.

```
def manejar_eventos(self, event):
    if event.type == pygame.MOUSEBUTTONDOWN:
        if event.button == 1:
            mouse_pos = event.pos
            if self.rect_single.collidepoint(mouse_pos):
                self.seleccion = 'SINGLE'
                return self.seleccion
            if self.rect_evolutivo.collidepoint(mouse_pos):
                self.seleccion = 'EVOLUTIVO'
                return self.seleccion

    if event.type == pygame.KEYDOWN:
        if event.key == pygame.K_1:
            self.seleccion = 'SINGLE'
            return self.seleccion
        elif event.key == pygame.K_2:
            self.seleccion = 'EVOLUTIVO'
            return self.seleccion
```

Módulo main.py (I)

El método `manejar_eventos()` devuelve el modo de juego cuando el usuario hace clic con el mouse sobre la opción correspondiente o al presionar las teclas 1/2.

```
def manejar_eventos(self, event):
    if event.type == pygame.MOUSEBUTTONDOWN:
        if event.button == 1:
            mouse_pos = event.pos
            if self.rect_single.collidepoint(mouse_pos):
                self.seleccion = 'SINGLE'
                return self.seleccion
            if self.rect_evolutivo.collidepoint(mouse_pos):
                self.seleccion = 'EVOLUTIVO'
                return self.seleccion

    if event.type == pygame.KEYDOWN:
        if event.key == pygame.K_1:
            self.seleccion = 'SINGLE'
            return self.seleccion
        elif event.key == pygame.K_2:
            self.seleccion = 'EVOLUTIVO'
            return self.seleccion
```

LALALALLALA