

Trabajo Práctico Final — Programación Computacional

Flappy Fish: Juego basado en Pygame

Julieta Zanoni, 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

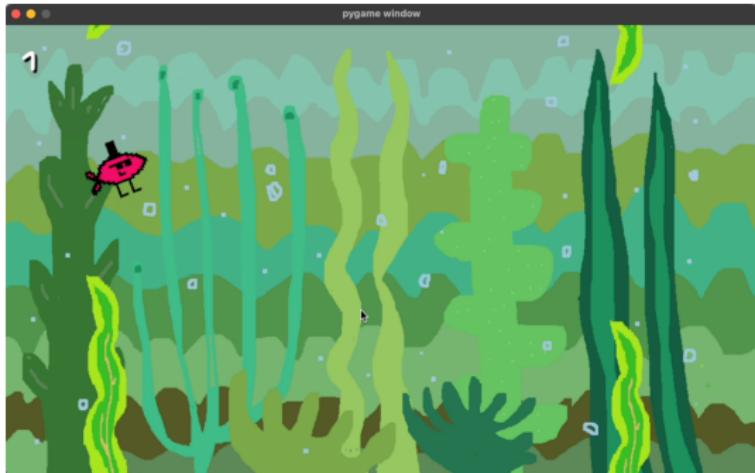
2 Arquitectura del Juego

Módulo game.py

Módulo fish.py

Introducción

Nuestro trabajo práctico está dividido en dos partes: trata sobre el desarrollo de un videojuego maunal inspirado en Flappy Bird, llamado Flappy Fish, implementado en Python utilizando la librería Pygame.



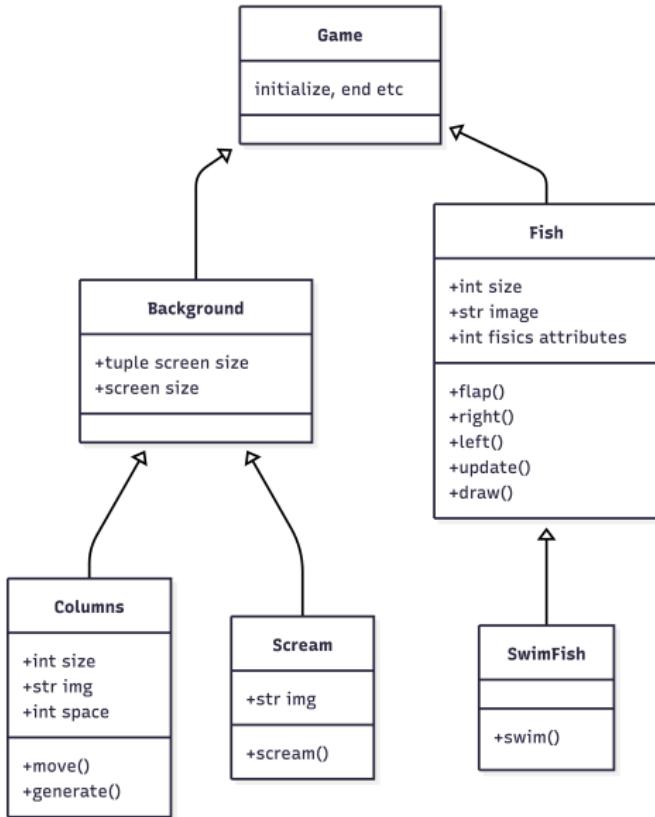
Introducción

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



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

Para entender mejor cómo estructurar el proyecto, 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.

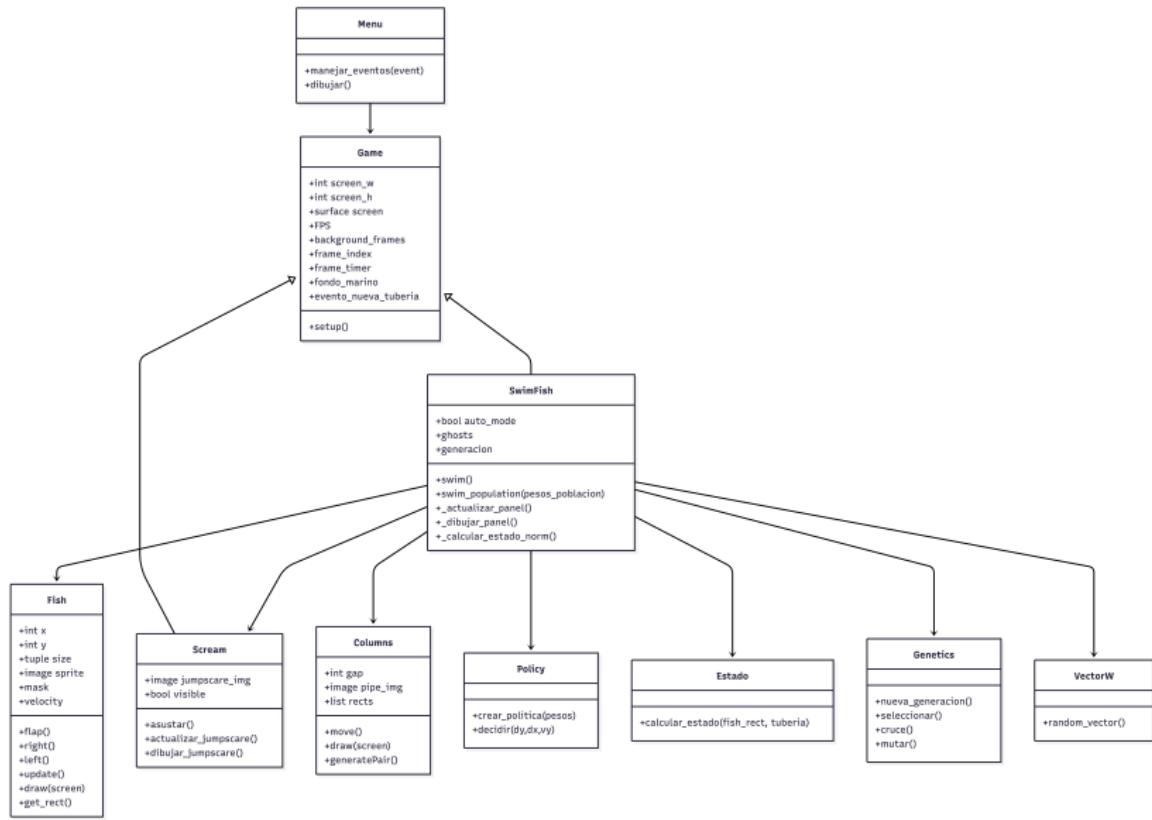
This screenshot shows the GitHub repository 'super-mario-python'. It displays a list of 266 commits from 52 contributors over 4 years. The commits are organized by file changes, with a focus on updates to classes, entities, img, levels, sfx, sprites, and traits. The repository has 10 branches and 0 tags. The commit history includes various bug fixes and improvements, such as reverting f-strings back to format(), adding attributes to entity classes, and fixing rendering issues.

File / Commit	Description	Date
reverted f-Strings back to format()	5 years ago	
Added attributes to entity class to make dealing with Koo...	4 years ago	
Revert *fix: Render coin animation with transparent backg...	5 years ago	
Allow mushroom to come out of box	5 years ago	
added sfx when big mario is damaged and fixed 2 bugs	5 years ago	
Update RedMushroom.json	5 years ago	
Fixed jumping in air bug	4 years ago	
Add more files to .gitignore	5 years ago	
Update README.md	5 years ago	
Fix Formatting	5 years ago	
reverted f-Strings back to format()	5 years ago	
Create requirements.txt	5 years ago	

This screenshot shows the GitHub repository 'Tower-Defense-Game'. It displays a list of 8 commits from 1 contributor over 6 years. The commits are organized by file changes, with a focus on adding files via upload for enemies, main_menu, menu, and towers. The repository has 1 branch and 0 tags. The commit history includes updates to README and game files, and the creation of .gitpod files.

File / Commit	Description	Date
Add files via upload	6 years ago	
Add files via upload	6 years ago	
Add files via upload	6 years ago	
Add files via upload	6 years ago	
Create .gitpod.dockerfile	6 years ago	
Create .gitpod.yml	6 years ago	
Updated README via script	last year	
Add files via upload	6 years ago	
Create requirements.txt	6 years ago	
Add files via upload	6 years ago	

Arquitectura del Juego



Arquitectura del Juego

El proyecto se estructura en los siguientes módulos:

- **game.py** — una clase de juego base.
- **fish.py** — física, movimiento y máscara del pez.
- **screamer** — el screamer para el juego en modo manual.
- **generacion_de_tuberias.py** — creación y movimiento de tuberías.
- **menu.py** — interfaz de menú.
- **swim_fish.py** — lógica manual y modo de Algoritmo Genético.
- **ml/** — política del agente, estado, 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, el 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: el 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: el 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: el 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 (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 (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 colisión 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)
```