



Dino v4.2.3

Manual del Programador

Alma Carena

Mateo Lugo

Dante Bassus

Santino Trevisano

Índice

Índice.....	1
1.Introducción.....	3
1.1 Propósito del documento.....	3
1.3 Alcance del proyecto.....	3
1.4 Público objetivo.....	3
2. Descripción general del sistema.....	4
2.1 Descripción del juego.....	4
2.2 Tecnologías y librerías utilizadas.....	4
2.3 Requerimientos.....	5
2.3.1 Software.....	5
2.3.2 Hardware.....	5
3. Arquitectura general del sistema.....	7
3.1 Flujo de comunicación entre Python y Arduino.....	7
3.2 Estructura de carpetas y archivos del proyecto.....	8
3.2.1 Función de cada archivo dentro del src.....	9
4. Diseño y lógica del juego.....	13
4.1 Idea y funcionamiento del juego.....	13
4.1.1 Flujo general:.....	13
4.1.2 Responsabilidades del bucle principal:.....	13
4.1.3 Integración hardware/software:.....	14
4.1.4 Persistencia y ranking:.....	14
4.1.5 Modularidad y buenas prácticas:.....	14
4.2 Controles y entradas.....	14
4.2.1 Entradas físicas:.....	14
4.2.2 Entradas digitales:.....	14
4.3 Salidas visuales y sonoras.....	15
4.3.1 Salidas visuales:.....	15
4.3.2 Salidas sonoras:.....	15
4.3.3 Gestión desde Python:.....	15
5.Módulos del sistema.....	16
5.1 Módulo idioma.....	16
5.5. Módulo elegir mundo.....	20
5.6. Módulo ver ranking.....	21
5.7. Modulo sonido.....	22
5.8. Módulo salir.....	23
5.9. Módulo principal.....	23
5.10 Módulo de comunicación serial (pyserial).....	23
5.11 Módulo de lógica del juego.....	24
6. Comunicación con Arduino.....	27
6.1 Descripción del protocolo serial.....	27
6.2 Configuración del puerto (baud rate, COM, timeout).....	27
6.3 Estructura de mensajes enviados y recibidos.....	27

6.3.1 Mensajes recibidos del Arduino:.....	27
6.3.2 Mensajes enviados al Arduino:.....	27
6.4 Ejemplo de comunicación (fragmento de código).....	28
7. Código.....	29
8. Créditos.....	30

1.Introducción

1.1 Propósito del documento

El presente manual tiene como propósito brindar una guía técnica detallada sobre el desarrollo, funcionamiento y mantenimiento del juego interactivo programado en Python y vinculado con una placa

1.2 Objetivo del sistema

El objetivo principal del sistema es crear un juego interactivo que combine la programación en Python con elementos físicos controlados mediante Arduino, generando una experiencia dinámica que integre el uso de luces y botones (touch) con una interfaz gráfica. El juego busca demostrar la comunicación serial entre el entorno digital y el hardware,

1.3 Alcance del proyecto

El proyecto abarca el desarrollo completo del software en Python y su vinculación con el hardware Arduino a través de comunicación serial. Incluye el diseño del código fuente, la estructura modular del sistema, la configuración del entorno de ejecución y las pruebas de funcionamiento. No se contempla en este alcance la implementación en dispositivos móviles. El sistema está diseñado para funcionar en computadoras con sistema operativo Windows, con una versión de Python 3.x instalada y con la librería pygame correctamente configurada.

1.4 Público objetivo

Este documento está dirigido a programadores, técnicos y estudiantes que deseen comprender la estructura del código, la comunicación entre hardware y software, y los procedimientos necesarios para ejecutar, modificar o ampliar el sistema.

El manual también sirve como referencia para futuras mejoras o adaptaciones del proyecto.

2. Descripción general del sistema

2.1 Descripción del juego

El programa que desarrollamos trata sobre un dino corriendo infinitamente mientras esquivando obstáculos, integramos diferentes librerías para realizar el juego e implementamos Mysql y Arduino para añadir más mecánicas y un mando físico.

2.2 Tecnologías y librerías utilizadas

Pygame : Es la librería principal que usamos para crear el juego. Gracias a Pygame pudimos manejar los gráficos, los sonidos y todo lo que el jugador hace con el teclado o el mouse. Básicamente, es la que nos permitió mostrar el personaje, los obstáculos y el fondo en pantalla.

Pyserial : Nos sirvió para conectar el juego hecho en Python con el Arduino. Con esta librería logramos que ambos se comuniquen por el puerto serial, así el Arduino puede enviar señales al juego o recibir información de él. Por ejemplo, sirve para que el personaje salte cuando tocamos un sensor físico.

Random : Se usa para generar números al azar. Con esto pudimos hacer que el juego no sea siempre igual, ya que permite que los obstáculos aparezcan en distintos momentos o posiciones. Así el juego se vuelve más variado y divertido.

Sys : Esta librería ayuda a que el programa se comuniquen con la computadora. En nuestro caso, la usamos para cerrar el juego de forma segura cuando termina o el jugador sale.

Os : Nos permite trabajar con archivos y carpetas dentro de la computadora. La usamos para poder encontrar fácilmente las imágenes o sonidos que usa el juego, sin importar en qué carpeta esté guardado el proyecto.

Time : La usamos para medir el tiempo y controlar la velocidad de ciertas acciones, como la espera para la conexión con Arduino o los intervalos entre eventos del juego.

Mysql.connector : Esta librería nos permitió conectar el juego con una base de datos MySQL. Así pudimos guardar y recuperar los puntajes de los jugadores, mostrando un ranking actualizado.

Serial.tools.list_ports : Es un submódulo de **pyserial** que nos ayuda a listar todos los puertos seriales disponibles en la computadora. Lo usamos para facilitar la detección del Arduino cuando está conectado.

Pymysql : Se utiliza para la conexión y manejo de la base de datos MySQL desde Python, permitiendo ejecutar consultas y gestionar los datos de los usuarios y puntajes.

2.3 Requerimientos

2.3.1 Software

1. Python 3.x instalado.
2. Librerías: pygame, pyserial, pymysql, pyinstaller.
3. MySQL Server y cliente configurado.
4. Arduino IDE y controladores.
5. Editor de código (VS Code o similar).
6. Inno Setup (opcional, para instalador).
7. Controladores USB para Arduino.
8. Recursos del juego (carpetas img, musica, etc.).

2.3.2 Hardware

1. PC con Windows 7 o superior.
2. Al menos 100 MB de espacio libre en disco.
3. Arduino (opcional, solo si se desea control del AQUILES):
 - a. Debe estar conectado al puerto COM4 (o el que corresponda en tu sistema).
 - b. Requiere cable USB y drivers instalados.
 - c. Resistencia de 220Ω x1
 - d. Protoboard y cables de conexión.
4. Altavoces o auriculares para música y efectos de sonido.

5. Teclado para controles estándar.

3. Arquitectura general del sistema

3.1 Flujo de comunicación entre Python y Arduino

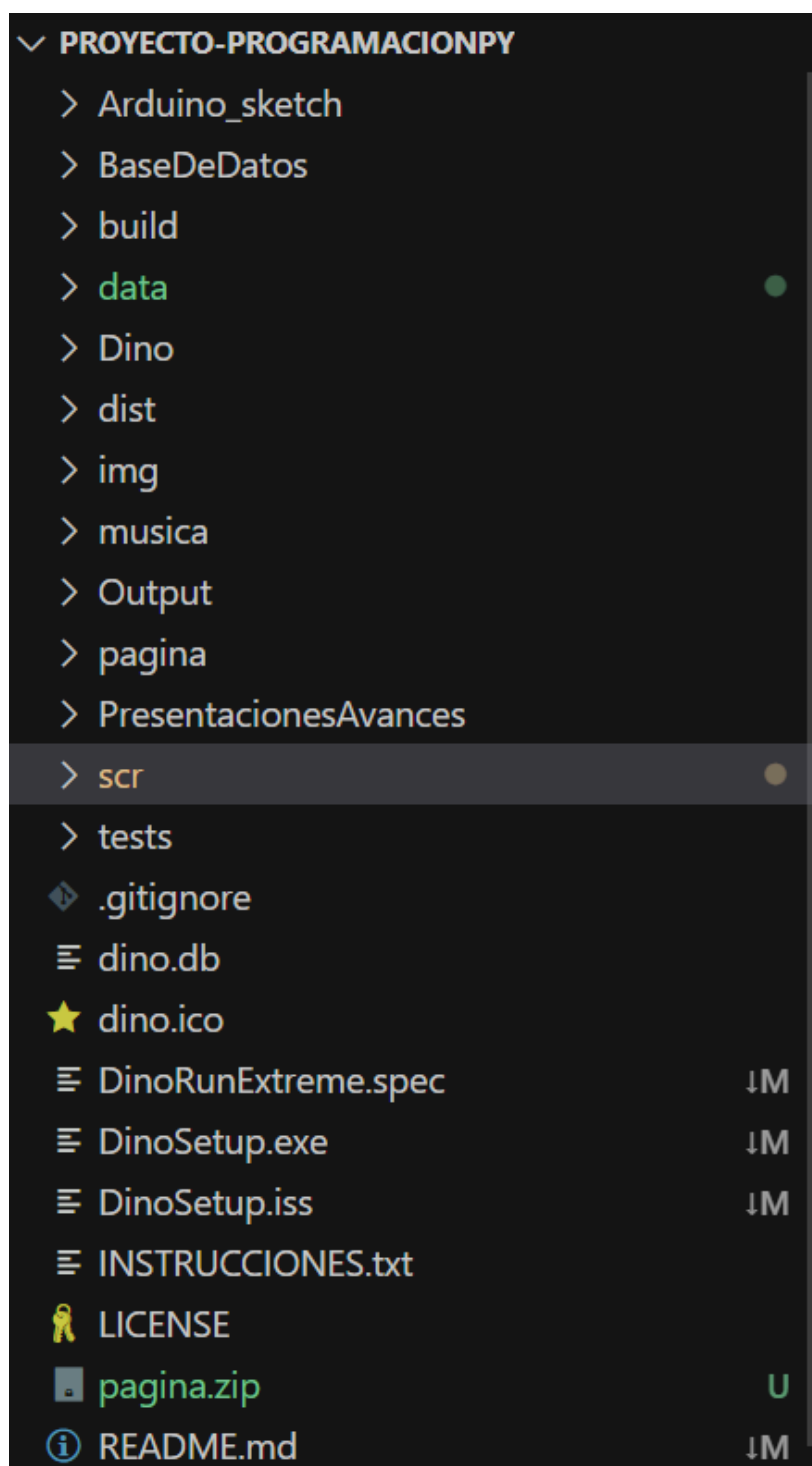
El sistema establece una integración directa entre el software desarrollado en Python y el hardware controlado por Arduino, permitiendo una experiencia de juego más dinámica y física.

La comunicación entre ambos se realiza mediante el puerto serial, utilizando la librería pyserial, la cual posibilita el envío y recepción de datos en tiempo real.

Por un lado, el programa en Python ejecuta el juego del dinosaurio a través de pygame, gestionando la interfaz visual, las colisiones, los sonidos, y la lógica principal del sistema. Además, se conecta a una base de datos MySQL, donde se almacenan los puntajes y registros de los jugadores.

Por otro lado, el Arduino cumple la función de mando físico. A través de botones (touch) conectados a la placa, el jugador puede realizar acciones dentro del juego, como saltar o agacharse, enviando señales que Python interpreta mediante la comunicación serial.

3.2 Estructura de carpetas y archivos del proyecto



Arduino_sketch : Contiene el código del arduino para comunicarse entre el programa y el hardware

BaseDeDatos : Contiene el código sql para la base de datos “dino”

Data/docs : Contiene los avances de los manuales de usuario,el manual del programador y el informe

PresentacionesAvances : Contiene todas las presentaciones con sus avances del proyecto que fuimos trabajando a lo largo del año

img : Contiene todos los archivos visuales utilizados en el juego

música : Contiene todos los archivos sonoros utilizados en el juego

página : Contiene el código HTML , CSS y SCRIPT .JS de nuestra página

src : Contiene todos los archivos python trabajados en el proyecto

test : Contiene todos los códigos de prueba

README : contiene información fundamental sobre el proyecto

3.2.1 Función de cada archivo dentro del src

Game_object.py : El archivo game_objects.py contiene la definición de las clases que representan los distintos elementos del juego. En este caso, se encuentran clases como Perro, Obstáculo y Fondo.

Cada clase posee atributos y métodos que encapsulan su comportamiento: el perro controla la animación y el salto, los obstáculos gestionan su desplazamiento y eliminación al salir de la pantalla, y el fondo administra el desplazamiento continuo que genera la sensación de movimiento.

main.py : El archivo main.py cumple el rol de punto de entrada del juego. Allí se configura la ventana principal, se cargan los recursos gráficos, se inicializan los objetos principales (como el personaje, el fondo y los obstáculos) y se ejecuta el bucle principal. Dicho bucle es el núcleo del juego, ya que gestiona la captura de eventos (como las teclas presionadas), la actualización de la lógica de los objetos (movimiento, animaciones y detección de colisiones) y la representación visual de todos los elementos en la pantalla.

utils.py : el archivo utils.py se utiliza como un conjunto de herramientas auxiliares. En él se definen funciones de uso común, como la que permite mostrar texto en pantalla. Estas funciones pueden emplearse en distintas partes del proyecto, evitando la repetición de código y contribuyendo a la reutilización y legibilidad del programa.

menu.py : El archivo *menu.py* contiene la lógica y el diseño del menú principal del juego. Desde aquí el jugador puede acceder a las distintas opciones, como iniciar partida, seleccionar personaje, mundo o sonido, y salir del juego. Su función principal es gestionar la navegación entre las diferentes pantallas del programa.

seleccion_personaje.py : En *seleccion_personaje.py* se define la pantalla donde el jugador elige el personaje que va a utilizar. Se muestran las distintas opciones disponibles y se guarda la selección para que el juego la utilice al comenzar la partida.

seleccion_mundo.py : El archivo *seleccion_mundo.py* permite al jugador elegir el entorno o escenario en el que se desarrollará el juego. Cada mundo puede tener un fondo o dificultad diferente, y la elección se aplica al iniciar el juego.

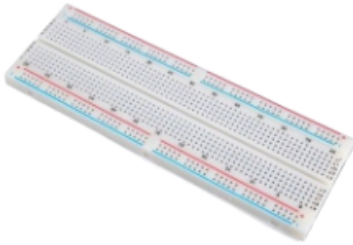
elegir_nombre.py : El archivo *elegir_nombre.py* gestiona la pantalla donde el jugador debe ingresar su nombre antes de comenzar la partida. Se encarga de mostrar el campo de texto, validar la entrada y devolver el nombre seleccionado, que luego será utilizado para registrar el puntaje o mostrarlo en el ranking.

mostrar_ranking.py : En *mostrar_ranking.py* se encuentra la lógica para desplegar la tabla de posiciones o ranking de jugadores. Este archivo se encarga de leer los puntajes almacenados, ordenarlos y presentarlos en pantalla de manera clara, permitiendo al jugador ver su posición respecto a los demás.

seleccionar_sonido.py : El archivo *seleccionar_sonido.py* permite al jugador elegir la música o los efectos de sonido que acompañarán la partida. Gestiona la lista de pistas disponibles y aplica la selección para personalizar la experiencia de juego.

Transiciones.py: El archivo *transiciones.py* contiene los elementos, formatos y valores de las transiciones del menú y del juego, algunas no se usaron pero están guardadas ahí, cuando son llamadas dependiendo de qué transición sea se ejecuta.

3.3 Descripción de cada componente



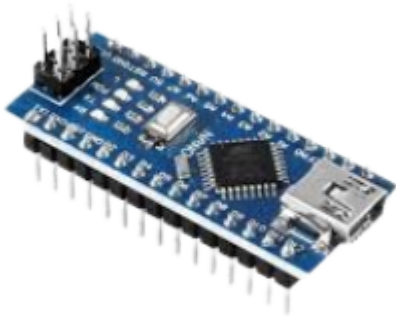
Protoboard: Se utilizó como principal componente, para poder realizar las conexiones con el cableado y los demás componentes



Resistencia de 220Ω x1: Se utilizó una resistencia de 220Ω para la luz led.



Botones pulsadores x4: Los botones pulsadores se usaron para poder interactuar con las funciones del juego.



Arduino Nano: El arduino se usó para poder programar con los diferentes pines, los botones y el led, para poder controlar el juego.



Cables: Se usaron para conectar todos los elementos en el protoboard.



Led x1: Se usó un led para poder tener una indicación de que los botones pulsadores funcionan en cada toque.

4. Diseño y lógica del juego

4.1 Idea y funcionamiento del juego

El juego implementa la mecánica clásica del “dinosaurio que corre infinitamente”, donde el objetivo principal es esquivar obstáculos y sobrevivir el mayor tiempo posible para alcanzar la máxima puntuación. El sistema está diseñado para ser modular, flexible y fácilmente ampliable, integrando tanto componentes de software como de hardware.

4.1.1 Flujo general:

Al iniciar, el usuario selecciona idioma, ingresa su nombre y configura la partida (personaje, mundo, música) mediante un menú principal.

El juego utiliza la librería Pygame para gestionar la ventana, renderizar gráficos, reproducir sonidos y capturar eventos de entrada.

El personaje (Perro o Gato) avanza automáticamente por el escenario, mientras el jugador controla sus acciones (saltar, agacharse) usando el teclado o, opcionalmente, botones físicos conectados a un Arduino.

Los obstáculos y enemigos se generan de forma aleatoria, aumentando la dificultad a medida que avanza el tiempo.

La velocidad del juego incrementa progresivamente, lo que exige mayor habilidad y reflejos del jugador.

Cada vez que el jugador supera un obstáculo, la puntuación aumenta. Al perder, el puntaje y el nombre se almacenan en una base de datos MySQL, permitiendo mantener un ranking histórico.

El sistema modular permite que cada pantalla y funcionalidad (menú, selección, ranking, lógica de juego, comunicación serial) esté implementada en archivos independientes, facilitando el mantenimiento y la extensión del código.

4.1.2 Responsabilidades del bucle principal:

Actualizar las posiciones del personaje y los obstáculos en cada frame.

Detectar colisiones entre el personaje y los obstáculos/enemigos.

Procesar entradas desde el teclado y desde Arduino (mediante pyserial).

Dibujar todos los elementos gráficos en pantalla, incluyendo animaciones, efectos y transiciones.

Reproducir sonidos y música según los eventos del juego.

Gestionar el flujo entre pantallas (inicio, menú, juego, ranking, salida).

4.1.3 Integración hardware/software:

El Arduino actúa como mando físico, enviando señales por puerto serial que el juego interpreta como eventos de control.

Un LED conectado al protoboard indica la activación de botones físicos.

El sistema está preparado para funcionar tanto con controles físicos como digitales, permitiendo una experiencia flexible.

4.1.4 Persistencia y ranking:

Al finalizar la partida, el sistema registra el nombre y puntaje del jugador en la base de datos.

El ranking puede consultarse desde el menú, mostrando los mejores resultados y fomentando la competencia.

4.1.5 Modularidad y buenas prácticas:

El código está organizado en módulos independientes, cada uno con responsabilidades claras.

Se emplean funciones y clases para encapsular la lógica y facilitar la reutilización.

La estructura de carpetas permite separar recursos gráficos, sonoros y de base de datos.

4.2 Controles y entradas

El sistema está diseñado para aceptar entradas tanto digitales (teclado) como físicas (botones conectados a Arduino), permitiendo una experiencia de juego flexible y ampliable.

4.2.1 Entradas físicas:

Touch (Botones):

Se utilizan botones pulsadores conectados al Arduino para controlar las acciones principales del juego. Cada botón está asociado a una función específica, como saltar, agacharse, moverse por el menú o confirmar selecciones. El Arduino detecta la pulsación y envía un mensaje por el puerto serial al programa en Python, que lo interpreta como un evento de control.

LED indicador:

Un LED conectado al protoboard se enciende cada vez que se detecta la pulsación de un botón físico. Esto proporciona retroalimentación visual inmediata al usuario, indicando que la entrada fue registrada correctamente.

4.2.2 Entradas digitales:

Teclado:

El juego también puede ser controlado completamente mediante el teclado. Las teclas de flecha (arriba, abajo, izquierda, derecha) y la tecla Enter permiten navegar por los menús, seleccionar opciones y controlar el personaje durante la partida.

4.3 Salidas visuales y sonoras

El sistema proporciona una experiencia audiovisual completa, combinando gráficos animados, efectos visuales y sonido dinámico para mejorar la inmersión del jugador.

4.3.1 Salidas visuales:

Movimiento fluido:

El personaje principal (Perro/Gato), los obstáculos y el fondo se animan de forma continua y suave utilizando la librería Pygame. Las animaciones de correr, saltar y agacharse se gestionan mediante sprites y ciclos de imágenes.

Actualización de puntuación:

La puntuación del jugador se muestra en tiempo real en la esquina superior de la pantalla, actualizándose cada vez que se supera un obstáculo.

Pantallas de estado:

El juego incluye una pantalla de inicio con opciones de configuración y una pantalla de “Game Over” que se muestra al perder, indicando el puntaje final y permitiendo reiniciar o salir.

Efectos visuales adicionales:

Se emplean transiciones, brillitos, fondos coloridos y medallas en el ranking para enriquecer la presentación gráfica.

4.3.2 Salidas sonoras:

Sonido de salto:

Cada vez que el personaje salta, se reproduce un efecto sonoro específico para dar retroalimentación inmediata al jugador.

Música de fondo:

El juego cuenta con música ambiental tanto en el menú como durante la partida, gestionada desde Python mediante Pygame. El usuario puede seleccionar la pista deseada antes de jugar.

Efectos en eventos clave:

Se reproducen sonidos adicionales en eventos como colisiones, selección de opciones y finalización de la partida.

4.3.3 Gestión desde Python:

Todos los sonidos y música se cargan y reproducen utilizando las funciones de Pygame (pygame.mixer).

El sistema permite controlar el volumen, pausar, detener y cambiar la música según el estado del juego

5. Módulos del sistema

5.1 Módulo idioma

Permite seleccionar y cambiar el idioma de la interfaz (Español/Inglés). Cada pantalla y mensaje utiliza diccionarios de textos según el idioma activo.

Interfaz:

- Variable global o atributo idioma en cada clase.
- Diccionario textos con claves para cada idioma.
- Métodos para cambiar idioma y actualizar textos en tiempo real.

```
self.textos = {"es": {...}, "en": {...}}
self.idioma = "es"
self.txt = self.textos[self.idioma]
```

5.2 Módulo elegir nombre

Pantalla donde el usuario selecciona su nombre (4 letras). Valida que el nombre no esté repetido en la base de datos y lo guarda.

- Método mostrar() para la interfaz gráfica y lógica de selección.
- Métodos para verificar duplicados (nombre_existe) y guardar el nombre (guardar_nombre).
- Interacción con la base de datos MySQL.

```
def mostrar(self):
    clock = pygame.time.Clock()

    while True:
        # Fondo colorido con estrellas
        self.dibujar_fondo_colorido()

        # Actualizar y dibujar brillitos
        self.actualizar_brillitos()

        # Título colorido
        # MODIFICADO: Usar self.txt y color dorado
        titulo_texto = self.txt["titulo"]
        sombra = self.fuente_titulo.render(titulo_texto, True, (100, 50, 0)) # Sombra dorada oscura
        texto = self.fuente_titulo.render(titulo_texto, True, self.color_titulo) # Dorado
        rect = texto.get_rect(center=(self.ancha // 2, 120))
        self.pantalla.blit(sombra, (rect.x + 3, rect.y + 3))
        self.pantalla.blit(texto, rect)
```

```

def guardar_nombre(self, nombre):
    """Guarda el nombre en la base de datos y devuelve su ID"""
    try:
        conexion = mysql.connector.connect(**self.db_config)
        cursor = conexion.cursor()
        cursor.execute("INSERT INTO usuario (Nombre) VALUES (%s);", (nombre,))
        conexion.commit()
        self.id_usuario_actual = cursor.lastrowid
        conexion.close()
        print(f"[INFO] Usuario '{nombre}' guardado con ID {self.id_usuario_actual}")
        return self.id_usuario_actual
    except Exception as e:
        print(f"[ERROR DB] {e}")
        return None # Retornar None en caso de error

```

```

def nombre_existe(self, nombre):
    """Verifica si un nombre ya existe en la base de datos"""
    try:
        conexion = mysql.connector.connect(**self.db_config)
        cursor = conexion.cursor()
        # Comparar en mayúsculas para evitar diferencias de case
        cursor.execute("SELECT COUNT(*) FROM usuario WHERE UPPER(Nombre) = %s", (nombre.upper(),))
        count = cursor.fetchone()[0]
        conexion.close()
        return count > 0
    except Exception as e:
        print(f"Error al verificar nombre: {e}")
        return False

```

5.3 Módulo menú

Menú principal con opciones: Jugar, Elegir Mundo, Elegir Personaje, Ver Ranking, Salir. Permite navegar con teclado o Arduino.

Interfaz:

- Método mostrar() para dibujar y gestionar el menú.
- Control de música, idioma y volumen.
- Devuelve la opción seleccionada para el flujo principal.

```
def mostrar(self):
    clock = pygame.time.Clock()

    while True:
        # Fondo colorido con estrellas
        self.dibujar_fondo_colorido()

        # Actualizar y dibujar brillitos
        self.actualizar_brillitos()

        # Título colorido
        # MODIFICADO: Usar self.txt y color dorado
        titulo_texto = self.txt["titulo"]
        sombra = self.fuente_titulo.render(titulo_texto, True, (100, 50, 0)) # Sombra dorada oscura
        texto = self.fuente_titulo.render(titulo_texto, True, self.color_titulo) # Dorado
        rect = texto.get_rect(center=(self.ancho // 2, 120))
        self.pantalla.blit(sombra, (rect.x + 3, rect.y + 3))
        self.pantalla.blit(texto, rect)

        # Letras coloridas
        inicio_x = self.ancho // 2 - 1.5 * 140
        pos_y_letras = self.alto // 2
        for i in range(4):
            letra = self.letras[self.indice[i]]

            # Colores vibrantes según si está activa o no
            if i == self.posicion_actual:
                color_letra = self.color_letra_activa # Cyan vibrante
                color_caja = self.color_casilla_activa # Rojo vibrante
            else:
                color_letra = self.color_letra_normal # Azul claro
                color_caja = self.color_casilla_normal # Azul oscuro

            surf = self.fuente_letra.render(letra, True, color_letra)
            rect = surf.get_rect(center=(inicio_x + i * 140, pos_y_letras))
```

5.4. Módulo Elegir Personaje

Pantalla para elegir entre Perro o Gato, con animaciones y efectos visuales.

Interfaz:

- Método mostrar() para la selección y confirmación.
- Navegación con flechas o Arduino.
- Devuelve el personaje elegido.

```
personaje = SeleccionPersonaje(...).mostrar()
```

```
class SeleccionPersonaje:
    # MODIFICADO: Añadido arduino_serial=None e idioma
    def __init__(self, pantalla, ancho, alto, arduino_serial=None, idioma="es"):
        self.pantalla = pantalla
        self.ancho = ancho
        self.alto = alto
        self.arduino_serial = arduino_serial # <-- MODIFICADO
        self.idioma = idioma

    # --- Textos Multi-idioma (MODIFICADO) ---
    self.textos = {
        "es": {
            "titulo": "Selecciona tu personaje",
            "perro": "Perro",
            "gato": "Gato",
            "volver": "VOLVER",
            "instruccion1": "Use 'Flecha Izquierda' para cambiar.",
            "instruccion2": "Use 'Flecha Abajo' para Volver. Use 'Flecha Derecha' para confirmar."
        },
        "en": {
            "titulo": "Select your character",
            "perro": "Dog",
            "gato": "Cat",
            "volver": "BACK",
            "instruccion1": "Use 'Left Arrow' to change.",
            "instruccion2": "Use 'Down Arrow' for Back. Use 'Right Arrow' to confirm."
        }
    }
    self.actualizar_textos() # Asegurar que los textos se carguen correctamente
```

```

if self.arduino_serial is not None and self.arduino_serial.is_open:
    try:
        while self.arduino_serial.in_waiting > 0:
            linea = self.arduino_serial.readline().decode('utf-8').strip()

            evento_tipo = None
            evento_key = None

            if linea == "UP_DOWN":
                evento_tipo = pygame.KEYDOWN
                evento_key = pygame.K_UP
            elif linea == "UP_UP":
                evento_tipo = pygame.KEYUP

```

5.5. Módulo elegir mundo

Permite elegir el escenario de juego: Día o Noche, mostrando fondos y efectos.

Interfaz:

- Método mostrar() para la selección.
- Devuelve el mundo elegido.

```

mundo = SeleccionMundo(...).mostrar()

```

```

class SeleccionMundo:
    # MODIFICADO: Añadido arduino_serial=None e idioma
    def __init__(self, ventana, ancho, alto, arduino_serial=None, idioma="es"):
        self.ventana = ventana
        self.ancho = ancho
        self.alto = alto
        self.arduino_serial = arduino_serial # <-- MODIFICADO
        self.idioma = idioma

    # --- Textos Multi-idioma (MODIFICADO) ---
    self.textos = {
        "es": {
            "titulo": "SELECCIONAR MUNDO",
            "noche": "NOCHE", # MODIFICADO
            "dia": "DÍA", # MODIFICADO
            "volver": "Volver",
            # MODIFICADO: Separado en dos claves
            "instruccion_cambiar": "Use 'Flecha Izquierda' para cambiar.",
            "instruccion_seleccionar": "Use 'Flecha Derecha' para seleccionar."
        },
        "en": {
            "titulo": "SELECT WORLD",
            "noche": "NIGHT", # MODIFICADO
            "dia": "DAY", # MODIFICADO
            "volver": "Back",
            # MODIFICADO: Separado en dos claves
            "instruccion_cambiar": "Use 'Left Arrow' to change.",
            "instruccion_seleccionar": "Use 'Right Arrow' to select."
        }
    }
    self.actualizar_textos() # Asegurar que los textos se carguen correctamente

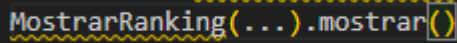
```

5.6. Módulo ver ranking

Muestra los mejores puntajes guardados en la base de datos, con medallas y nombres.

Base de datos:

- Tablas: usuario (nombre, id), ranking (puntaje, id_usuario).
- Consulta SQL para obtener el top 3.



MostrarRanking(...).mostrar()

```
def obtener_top3(self):
    print(f"[DEBUG] obtener_top3 llamado. cursor existe: {self.cursor is not None}")
    if not self.cursor:
        print("[DEBUG] No hay cursor, retornando lista vacía")
        write_log("[DEBUG] obtener_top3 llamado pero no hay cursor, lista vacía")
        return []
    try:
        self.cursor.execute("""
            SELECT u.Nombre, r.Puntaje
            FROM ranking r
            JOIN usuario u ON r.Id_Usuario = u.Id_Usuario
            ORDER BY r.Puntaje DESC
            LIMIT 3
        """)
        resultados = self.cursor.fetchall()
        print(f"[DEBUG] Resultados obtenidos: {resultados}")
        write_log(f"[DEBUG] Resultados obtenidos: {resultados}")
        return resultados
    except Exception as e:
        print(f"[ERROR SQL] {e}")
        write_log(f"[ERROR SQL] {e}")
        return []
```

5.7. Modulo sonido

Gestiona la música y efectos de sonido en el menú y el juego.

- Métodos: cargar_musica_menu(), iniciar_musica_menu(), pausar_musica_menu(), detener_musica_menu().
- Control de volumen y mute/unmute.

```
def desmutear_musica_menu(volumen):
    """Desmutea la música del menú"""
    try:
        volumen_musica = volumen * 0.4 # Restaurar volumen proporcional
        pygame.mixer.music.set_volume(volumen_musica)
        print(f"[MÚSICA MENÚ] Música desmuteada, volumen: {volumen_musica:.2f}")
    except Exception as e:
        print(f"[ERROR MÚSICA MENÚ] No se pudo desmutear la música: {e}")

def iniciar_musica_menu():
    """Inicia la música del menú en loop"""
    try:
        pygame.mixer.music.play(-1) # -1 significa loop infinito
        print("[MÚSICA MENÚ] Música del menú iniciada")
    except Exception as e:
        print(f"[ERROR MÚSICA MENÚ] No se pudo iniciar la música del menú: {e}")

def detener_musica_menu():
    """Detiene la música del menú"""
    try:
        pygame.mixer.music.stop()
        print("[MÚSICA MENÚ] Música del menú detenida")
    except Exception as e:
        print(f"[ERROR MÚSICA MENÚ] No se pudo detener la música del menú: {e}")

def pausar_musica_menu():
    """Pausa la música del menú"""
    try:
        pygame.mixer.music.pause()
        print("[MÚSICA MENÚ] Música del menú pausada")
    except Exception as e:
        print(f"[ERROR MÚSICA MENÚ] No se pudo pausar la música del menú: {e}")
```

5.8. Módulo salir

Permite salir del juego de forma segura, cerrando conexiones y recursos.

- Opción en el menú principal.
- Lógica para cerrar el programa y liberar recursos.

```
if opcion == "salir":
    pygame.quit()
    sys.exit()
```

5.9. Módulo principal

Controla el flujo completo del juego, llamando a los demás módulos en orden lógico.

- Función principal (main()) que gestiona la secuencia: intro, idioma, nombre, menú, submenús, juego, ranking, salida.

```
def main():
    mostrar_intro_epica()
    idioma = seleccionar_idioma_inicial()
    nombre, id_usuario = ElegirNombre(...).mostrar()
    while True:
        opcion = Menu(...).mostrar()
        # ...flujo según opción...
```

5.10 Módulo de comunicación serial (pyserial)

Recibe eventos de botones físicos desde Arduino por puerto serial, traduciéndolos a eventos de teclado en el juego.

- Inicialización del puerto: serial.Serial(port="COM3", baudrate=9600, timeout=0.1)
- Lectura de mensajes y conversión a eventos Pygame

```
Users > lucky > AppData > Local > Packages > PythonSoftwareFoundation.Python.3.13_qbz5n2kfra8p
#!/usr/bin/env python
#
# This is a wrapper module for different platform implementations
#
# This file is part of pySerial. https://github.com/pyserial/pyserial
# (C) 2001-2020 Chris Liechti <cliechti@gmx.net>
#
# SPDX-License-Identifier: BSD-3-Clause
```

```
if arduino_serial.in_waiting > 0:
    linea = arduino_serial.readline().decode('utf-8').strip()
    # Procesar evento
```

5.11 Módulo de lógica del juego

Define los objetos y reglas del juego: jugador, obstáculos, aves, fondo, colisiones y mecánicas.

- Clases: Perro, Obstaculo, Ave, Fondo.
- Métodos: update(), dibujar(), manejar_salto(), manejar_agacharse().
- Integración con el bucle principal para gestionar el estado y las interacciones.

```
class Perro(pygame.sprite.Sprite):
    # --- MODIFICADO: __init__ (Se quitó el Dash) ---
    def __init__(self, frames_correr, imagen_salto, imagen_aire, ancho, alto, altura_suelo):
        super().__init__()
        self.frames_correr = frames_correr
        self.imagen_salto = imagen_salto
        self.imagen_aire = imagen_aire
        self.image = frames_correr[0]
        self.rect = self.image.get_rect()
        self.rect.bottom = alto - altura_suelo
        self.rect.left = 50

        self.altura_suelo = altura_suelo
        self.alto_pantalla = alto
        self.vel_y = 0
        self.gravedad = 0.8 # REDUCIDO: Gravedad más suave (era 1.0)
        self.en_suelo = True
        self.frame_index = 0
        self.tiempo_anim = 0
        # Agachado
        self.is_crouching = False
        # Tecla abajo presionada
        self.down_pressed = False
        # Tecla espacio presionada
        self.space_pressed = False
        # Agachado en aire (flotar)
        self.air_crouch = False
```

```
# =====
class Obstaculo(pygame.sprite.Sprite):
    def __init__(self, imagenes, ancho, alto, altura_suelo, velocidad):
        super().__init__()
        self.image = random.choice(imagenes)
        self.rect = self.image.get_rect()
        self.rect.bottom = alto - altura_suelo
        self.rect.left = ancho + random.randint(0, 50) # Menos aleatoriedad

        # MODIFICADO: Se quitó _base_vel, ya no es necesario
        self.velocidad = velocidad
        self.mask = pygame.mask.from_surface(self.image)

    def update(self):
        self.rect.x -= self.velocidad
        if self.rect.right < 0:
            self.kill()
```

```

class Ave(pygame.sprite.Sprite):
    def __init__(self, imagenes, ancho_ventana, alto_ventana, velocidad_juego):
        super().__init__()

        self.imagenes = imagenes
        self.indice_animacion = 0
        self.image = self.imagenes[self.indice_animacion]
        self.tiempo_animacion = 0
        self.velocidad_animacion = 150

        # --- Posición y movimiento ---
        self.rect = self.image.get_rect()
        self.rect.x = ancho_ventana
        # MODIFICADO: Rango de altura de vuelo
        self.rect.y = random.choice([alto_ventana - 180, alto_ventana - 250, alto_ventana - 320])
        self.velocidad = velocidad_juego + random.uniform(1.5, 3.0) # Velocidad variable
        # MODIFICADO: Se quitó _base_vel
        self.mask = pygame.mask.from_surface(self.image)
        self.ultimo_update = pygame.time.get_ticks()

    def update(self):
        # Mover el ave hacia la izquierda
        self.rect.x -= self.velocidad

        # Animar el ave
        ahora = pygame.time.get_ticks()
        if ahora - self.ultimo_update > self.velocidad_animacion:
            self.ultimo_update = ahora
            self.indice_animacion = (self.indice_animacion + 1) % len(self.imagenes)
            self.image = self.imagenes[self.indice_animacion]
            # Es importante actualizar la máscara si la imagen cambia
            self.mask = pygame.mask.from_surface(self.image)

        # Eliminar el sprite si sale de la pantalla
        if self.rect.right < 0:
            self.kill()

```

```

class Fondo:
    def __init__(self, imagen, velocidad):
        self.imagen = imagen
        self.velocidad = velocidad
        self.x1 = 0
        self.x2 = imagen.get_width()

    def actualizar(self, velocidad_juego):
        # MODIFICADO: Se quitó la multiplicación por dash
        self.x1 -= velocidad_juego * self.velocidad
        self.x2 -= velocidad_juego * self.velocidad

        if self.x1 + self.imagen.get_width() < 0:
            self.x1 = self.x2 + self.imagen.get_width()
        if self.x2 + self.imagen.get_width() < 0:
            self.x2 = self.x1 + self.imagen.get_width()

    def dibujar(self, pantalla):
        pantalla.blit(self.imagen, (self.x1, 0))
        pantalla.blit(self.imagen, (self.x2, 0))

```

```

if juego_activo and juego_iniciado: # MODIFICADO: Solo ejecutar lógica del juego si ya empezó
    # MODIFICADO: Se quitó toda la lógica de DASH
    fondo.actualizar(velocidad_juego)
    jugador.actualizar(dt)
    obstaculos.update()
    aves.update()

    tiempo_actual = pygame.time.get_ticks()

    if tiempo_actual - tiempo_ultima_ave > intervalo_ave:
        aves.add(Ave(ave_imgs, ANCHO, ALTO, velocidad_juego))
        tiempo_ultima_ave = tiempo_actual
        intervalo_ave = random.randint(5000, 9000)

# --- LÓGICA DE OBSTÁCULOS MODIFICADA ---
if tiempo_actual - tiempo_ultimo_obstaculo > intervalo_cactus:

    # Decidir si generar uno o dos
    if random.random() < CHANCE_DOBLE_CACTUS:
        # Generar DOS cactus
        separacion = random.randint(SEPARACION_MIN, SEPARACION_MAX)
        obstaculos.add(Obstaculo(cactus_imgs, ANCHO, ALTO, ALTURA_SUELO, velocidad_juego))
        # El segundo puede ser pequeño
        obstaculos.add(Obstaculo(cactus_small, ANCHO + cactus_imgs[0].get_width() + separacion, ALTO, ALTURA_SUELO, velocidad_
    else:
        # Generar UN cactus
        obstaculos.add(Obstaculo(cactus_imgs, ANCHO, ALTO, ALTURA_SUELO, velocidad_juego))

# Evitar que un ave aparezca justo después de un cactus
tiempo_actual_local = pygame.time.get_ticks()
tiempo_desde_ultima_ave = tiempo_actual_local - tiempo_ultima_ave
if intervalo_ave - tiempo_desde_ultima_ave < 1000: # Si falta menos de 1s para un ave
    tiempo_ultima_ave = tiempo_actual_local # Reiniciar timer del ave

```

6. Comunicación con Arduino

6.1 Descripción del protocolo serial

El juego se comunica con Arduino usando el puerto serial. El Arduino envía mensajes de texto simples que representan eventos de botones (por ejemplo, "UP_DOWN", "LEFT_DOWN"). Cada mensaje corresponde a una acción en el juego (moverse por el menú, saltar, agacharse, etc.).

6.2 Configuración del puerto (baud rate, COM, timeout)

La configuración típica del puerto serial en el código es:

- Baud rate: 9600
- Puerto COM: Depende del sistema, lo busca y selecciona el juego automáticamente. (ejemplo: "COM3" en Windows)
- Timeout: 0.1 segundos (para evitar bloqueos)

```
import serial
arduino_serial = serial.Serial(port="COM3", baudrate=9600, timeout=0.1)
```

6.3 Estructura de mensajes enviados y recibidos.

6.3.1 Mensajes recibidos del Arduino:

"UP_DOWN": Flecha arriba presionada

"DOWN_DOWN": Flecha abajo presionada

"LEFT_DOWN": Flecha izquierda presionada

"RIGHT_DOWN": Flecha derecha presionada

6.3.2 Mensajes enviados al Arduino:

El juego normalmente solo recibe mensajes; no envía comandos al Arduino.

6.4 Ejemplo de comunicación (fragmento de código)

```
if arduino_serial is not None and arduino_serial.is_open:
    while arduino_serial.in_waiting > 0:
        linea = arduino_serial.readline().decode('utf-8').strip()
        evento_tipo = None
        evento_key = None

        if linea == "UP_DOWN":
            evento_tipo = pygame.KEYDOWN
            evento_key = pygame.K_UP
        elif linea == "DOWN_DOWN":
            evento_tipo = pygame.KEYDOWN
            evento_key = pygame.K_DOWN
        elif linea == "LEFT_DOWN":
            evento_tipo = pygame.KEYDOWN
            evento_key = pygame.K_LEFT
        elif linea == "RIGHT_DOWN":
            evento_tipo = pygame.KEYDOWN
            evento_key = pygame.K_RIGHT

        if evento_tipo is not None:
            evento_post = pygame.event.Event(evento_tipo, key=evento_key)
            pygame.event.post(evento_post)
```

7. Código

Si quieres ver mas detalle del código, ingrese [aquí](#)

Puedes hacer git clone para descargar el proyecto y trabajar directamente desde ahí! puedes contactarnos por gmail o github para poder subir commits si quieres o etc.

8. Créditos

Desarrolladores:

- Alma Carena
- Santino Trevisano
- Facundo Noriega
- Matero Lugo
- Dante Bassus

Docentes:

- Yamil Ganduglia
- York Mansilla