1942

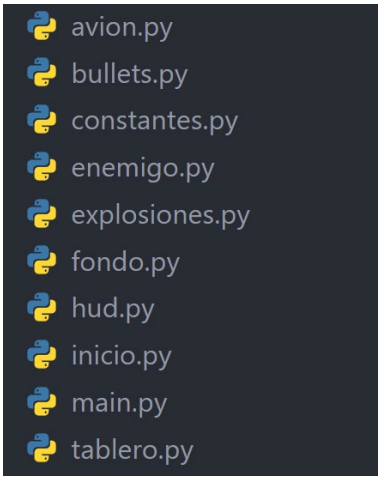
Autores:
Mario Ramos Salsón
Miguel Fidalgo García

10/10/22 **Grupo 81**

ÍNDICE:

1. Clases y explicación de los atributos y métodos principales.....3-5
2. Descripción general de los algoritmos utilizados para resolver los problema.....6
3. Problemas que nos hemos encontrado a lo largo del desarrollo 6
4. Funcionalidades extra incluidas en el juego.....7
5. Conclusiones 7

Clases y explicación de los atributos y métodos principales:



A lo largo de todo el proyecto hemos hecho uso de 10 clases, para manejar los diferentes objetos del juego, junto con sus funcionalidades, método y atributos.

Cabe destacar que a pesar de que la mayoría de las clases manejan objetos, como puede ser el caso de la clase avión o enemigo entre otras. El fichero main.py y el tablero.py manejan el lanzamiento del juego y todo el comportamiento del mismo, respectivamente.

→ Avión

En la clase avión, hemos configurado todo el movimiento del avión, junto con el giro de las hélices y la voltereta que puede llegar a ejecutar tres veces por partida, la cual hace que consigamos esquivar una bala y a los enemigos, para no perder una vida. Además, también hemos establecido sus atributos principales, como puede ser su coordenada x e y, su velocidad en ambas direcciones, su “sprite", junto con el tamaño de su altura y de su anchura. También, cabe destacar, que hemos hecho uso de getters y setters, para encapsular los atributos del objeto.

```python
def move(self, direction:str, size_x:int, size_y:int):

    if direction.lower() == "right" and self.x  <= size_x - self.plane_size_x:

        self.x += (1 * self.speed_x)

    if direction.lower() == "left" and self.x > 0:

        self.x -= (1 * self.speed_x)

    if direction.lower() == "up" and self.y > 0:

        self.y -= (1 * self.speed_y)

    if direction.lower() == "down" and self.y <= size_y - self.plane_size_y:

        self.y += (1 * self.speed_y)
```

Este método, configura el movimiento del avión, tanto hacia arriba, hacia abajo, izquierda y derecha como en diagonal en todas las direcciones.

```python
def helices_loop(self):

    if self.loop_bool == False:
        if pyxel.frame_count % 3 == 0 :

            self.sprite = constantes.AVION_SPRITE_1

        else:
            self.sprite = constantes.AVION_SPRITE_2

    else:
        if (pyxel.frame_count - self.frame0) % 25 < 5:
            self.sprite = constantes.AVION_LOOP[0]
        elif 5 <= (pyxel.frame_count - self.frame0) % 25 < 10:
            self.sprite = constantes.AVION_LOOP[1]
        elif 10 <= (pyxel.frame_count - self.frame0) % 25 < 15:
            self.sprite = constantes.AVION_LOOP[2]
        elif 15 <= (pyxel.frame_count - self.frame0) % 25 < 20:
            self.sprite = constantes.AVION_LOOP[3]
        elif 20 <= (pyxel.frame_count - self.frame0) % 25 < 25:
            self.loop_bool = False
```

Este método, configura tanto el movimiento de las hélices, como la voltereta explicada anteriormente.

→ Enemigos

En la clase de los enemigos, hemos implementado todas las características que cumplen los enemigos. Dando atributos específicos a algunos enemigos, como es el caso del regular, que tiene un atributo que determina si ya se ha dado la vuelta o no, para que así no dispare cuando está subiendo. Otro ejemplo pueden ser las coordenadas x e y de cada enemigo, ya que, el regular, el bombardero o el super bombardero pueden aparecer tanto por arriba como por abajo. Sin embargo, el enemigo rojo aparece por la izquierda. Además, también hemos establecido las diferentes puntuaciones que dan los enemigos una vez destruidos, junto con su vida.

→ Constantes

En la clase de las constantes, tenemos todo lo relacionado con aquel código, que aporta información permanente, como pueden ser los “sprites” de los enemigos o del avión. También hemos añadido cada una de las velocidades dependiendo del tipo de bala, o por ejemplo el retardo que tiene el enemigo al disparar (llamado cooldown). Cabe destacar que este fichero, a pesar de no ser un objeto, tiene una gran utilidad, ya que, por ejemplo, si queremos cambiar la velocidad del avión, con solo cambiar el valor establecido en este fichero, ya cambia en todo el código.

→ Balas

En esta clase, hemos configurado el movimiento de las balas, de los tres tipos de objetos diferentes que pueden disparar. Estos tres son, el avión, el enemigo regular y el enemigo bombardero.

```
def move(self):

    if self.tipo == "AVION":
        self.y -= 1 * constantes.VELOCIDAD_BALAS

    if self.tipo == "REGULAR":
        self.y += (1 * constantes.VELOCIDAD_BALAS_ENEMIGO_REGULAR)

    if self.tipo == "BOMBARDERO":

        if self.pos == 0:

            self.y += (1 * constantes.VELOCIDAD_BALAS_ENEMIGO_BOMBARDERO)

        else:

            self.y -= (1 * constantes.VELOCIDAD_BALAS_ENEMIGO_BOMBARDERO)
```

→ Fondo

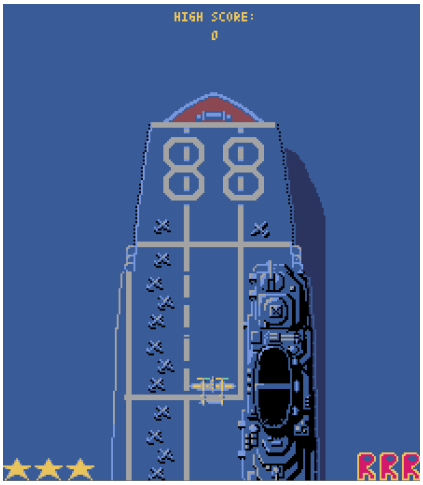
En este fichero, hemos establecido todas las posiciones y “sprites” de los textos e imágenes que hemos utilizado en la pantalla de inicio, donde esperamos a que el jugar pulse el ESPACIO para empezar a jugar.

```
if self.tipo == "PORTAVIONES":
    self.sprite = constantes.SPRITE_PORTAVIONES
    self.x = 51.5

if self.tipo == "ISLA":

    """Aquí elegimos entre la isla 1 y la 2"""

    self.isla = random.randint(0 , len(constantes.SPRITE_ISLA) - 1)

    """Aquí seleccionamos el sprite de la isla aleatoria obtenida"""

    self.sprite = constantes.SPRITE_ISLA[self.isla]
    self.x = random.randint(-(self.sprite[3]//2), (constantes.ANCHO - self.sprite[3]//
    self.y = -(self.sprite[4])
```

→ HUD

En esta clase, hemos implementado, todo lo que necesita la interfaz gráfica para funcionar, como el contador de vidas que le quedan al avión o las volteretas restantes que puede hacer. Todo esto lo hemos implementado en un método.

```
def draw(self):

    """Todos los sprites de las vidas"""

    pyxel.blt(0, constantes.ALTO - self.sprite_estrella[4]
              , *self.lista_estrella[0], colkey=0)

    pyxel.blt(self.sprite_estrella[3] + 2, constantes.ALTO - self.sprite_estrella[4],
              *self.lista_estrella[1], colkey=0)

    pyxel.blt((self.sprite_estrella[3] + 2)*2, constantes.ALTO - self.sprite_estrella[4],
              *self.lista_estrella[2], colkey=0)

    """Todos los sprites de los loops"""

    pyxel.blt(constantes.ANCHO - (self.sprite_loop[3] * 3) - 4,
              constantes.ALTO - self.sprite_loop[4],
              *self.lista_loop[0], colkey=7)

    pyxel.blt(constantes.ANCHO - (self.sprite_loop[3] * 2) - 2,
              constantes.ALTO - self.sprite_loop[4],
              *self.lista_loop[1], colkey=7)

    pyxel.blt(constantes.ANCHO - (self.sprite_loop[3]),
              constantes.ALTO - self.sprite_loop[4],
              *self.lista_loop[2], colkey=7)
```



→ Tablero

Esta clase es la más importante de todas, ya que aquí, se configura todo el juego, y se importan todas las clases mencionadas anteriormente. Además, se crean todas las listas necesarias para controlar cosas como las explosiones, los enemigos, las balas, las vidas o los loops.

Descripción de los algoritmos utilizados para resolver los problemas

Entre los algoritmos más útiles que hemos usado, para resolver los diferentes problemas, encontramos el uso de distancias para comprobar las colisiones, junto con la evaluación constante de las coordenadas de los enemigos junto con la de las balas, para detectar la colisión entre ambos.

Este código evalúa la posición del enemigo y de la bala, para ver cuándo ambos colisionan, y así eliminar ambos objetos de su lista correspondiente.

```
if enemigo.x < bala.x < enemigo.x + enemigo.sprite[3] + bala.sprite[3]\
and enemigo.y < bala.y < enemigo.y + enemigo.sprite[4]:
```

Este algoritmo calcula la distancia entre los enemigos y el avión en todo momento, para que cuando esta sea 0, detecte la colisión y ambos objetos se eliminen. Cabe destacar que el avión, solo se eliminará cuando pierda sus tres vidas.

```
r_enemigo, r_avion = enemigo.sprite[4]/2, self.avion.sprite[4]/2
dif_x = ((self.avion.x + self.avion.sprite[3]/2) - (enemigo.x + enemigo.sprite[3]/2))**2
dif_y = ((self.avion.y + self.avion.sprite[4]/2) - (enemigo.y + enemigo.sprite[4]/2))**2
distancia = math.sqrt(dif_x + dif_y)
```

Por otra parte, hemos seleccionado un movimiento para el juego peculiar. Haciendo uso de la “w” para ir hacia arriba, la “d” para ir a la derecha, “s” hacia abajo, “a” para la izquierda y el espacio para hacer el loop. Esto no es algo aleatorio, sino que esta distribución es la que se suele utilizar en el mundo de los videojuegos.

```
if pyxel.btn(pyxel.KEY_W):
    self.avion.move("up", self.width, self.heigth)
if pyxel.btn(pyxel.KEY_S):
    self.avion.move("down", self.width, self.heigth)
if pyxel.btn(pyxel.KEY_A):
    self.avion.move("left", self.width, self.heigth)
if pyxel.btn(pyxel.KEY_D):
    self.avion.move("right", self.width, self.heigth)
```

Además, hemos decidido hacer que el juego termine una vez llegado a los 1000 puntos, en vez de depender el número de enemigos que salgan.

Problemas que nos hemos encontrado a lo largo del desarrollo

Entre los problemas más notorios con los que nos hemos encontrado a lo largo de todo el desarrollo del juego, están: la falta de conocimientos acerca de cómo programar un juego, la gran dificultad para entender el funcionamiento de la librería "Pyxel" (ya que encontrábamos muy difícil la lectura de la documentación que había en GitHub), y por último también hemos notado la presión por la falta de tiempo (ya que la entrega coincidía con muchas otras actividades y exámenes establecidos por la universidad, además sólo hemos contado con tres semanas para aprender, entender, programar, diseñar, pulir y rectificar errores, para que el código fuese el más adecuado posible).

Funcionalidades extra incluidas en el juego

Pese a que no hemos optado por incluir grandes funciones adicionales a nuestro juego, sí que merecen ser destacados un par de detalles que ayudan a que este sea dinámico y no resulte excesivamente complicado. Estas son la adición de un loop extra al eliminar a un enemigo de tipo bombardero y la obtención de una vida extra cuando se acaba con un superbombardero. Es por ello que la frecuencia del superbombardero es baja y aparece cada cierto tiempo ya que entendemos que, de otra forma haría el juego mucho más sencillo.

Conclusiones

No obstante, encontramos este trabajo muy divertido de realizar, y muy útil a la hora de aprender cómo funciona el paradigma de la programación orientada objetos, junto con el uso de clases, métodos, atributos y funcionalidades como la encapsulación, los setters o los getters.

Personalmente creemos que hemos aprendido bastante y hemos formado una buena base para futuros trabajos relacionados con todo esto, pero cabe destacar que hemos notado una fuerte carencia de información en internet, que en ciertas ocasiones ha hecho que nos cueste mucho solucionar algunos problemas.

Por último, creemos que se podría mejorar algunas de las explicaciones en clase, ya que consideramos que el uso del "tilemap" nos hubiera ayudado mucho a organizar todo el fondo, no obstante, al final tampoco hemos llegado a notar esto como un problema demasiado grande ya que conseguimos resolverlo con las explicaciones dadas en clase.