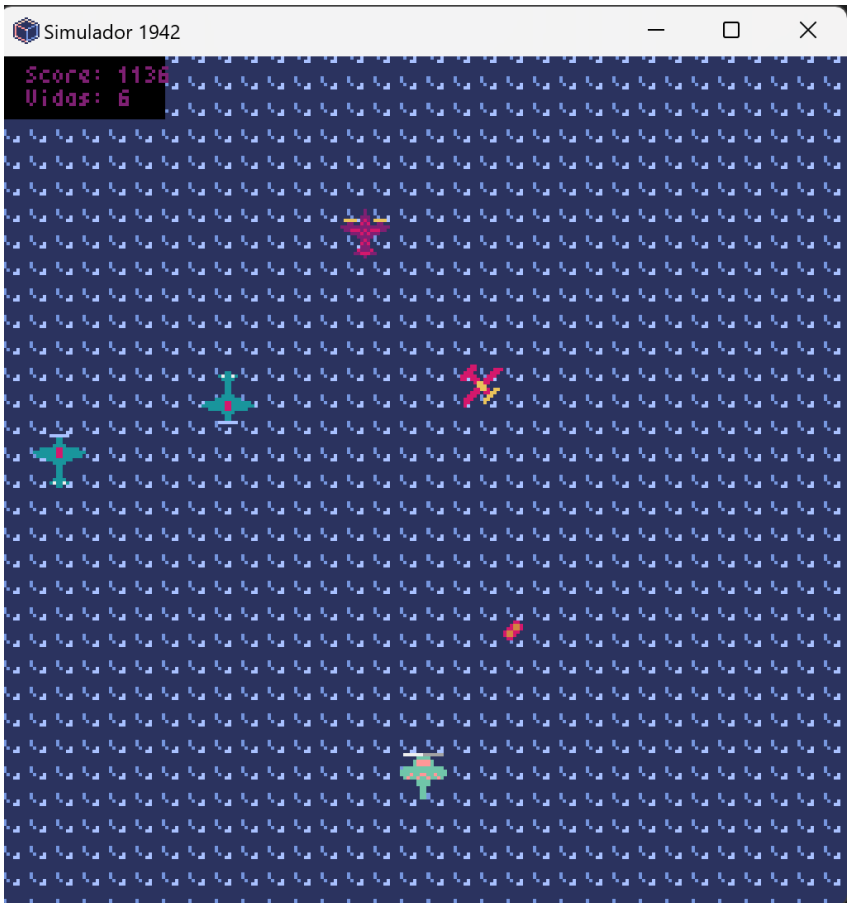
Memoria

1942: Simulador



Autores

Eoin O’Hearn y José de Viana

Introducción

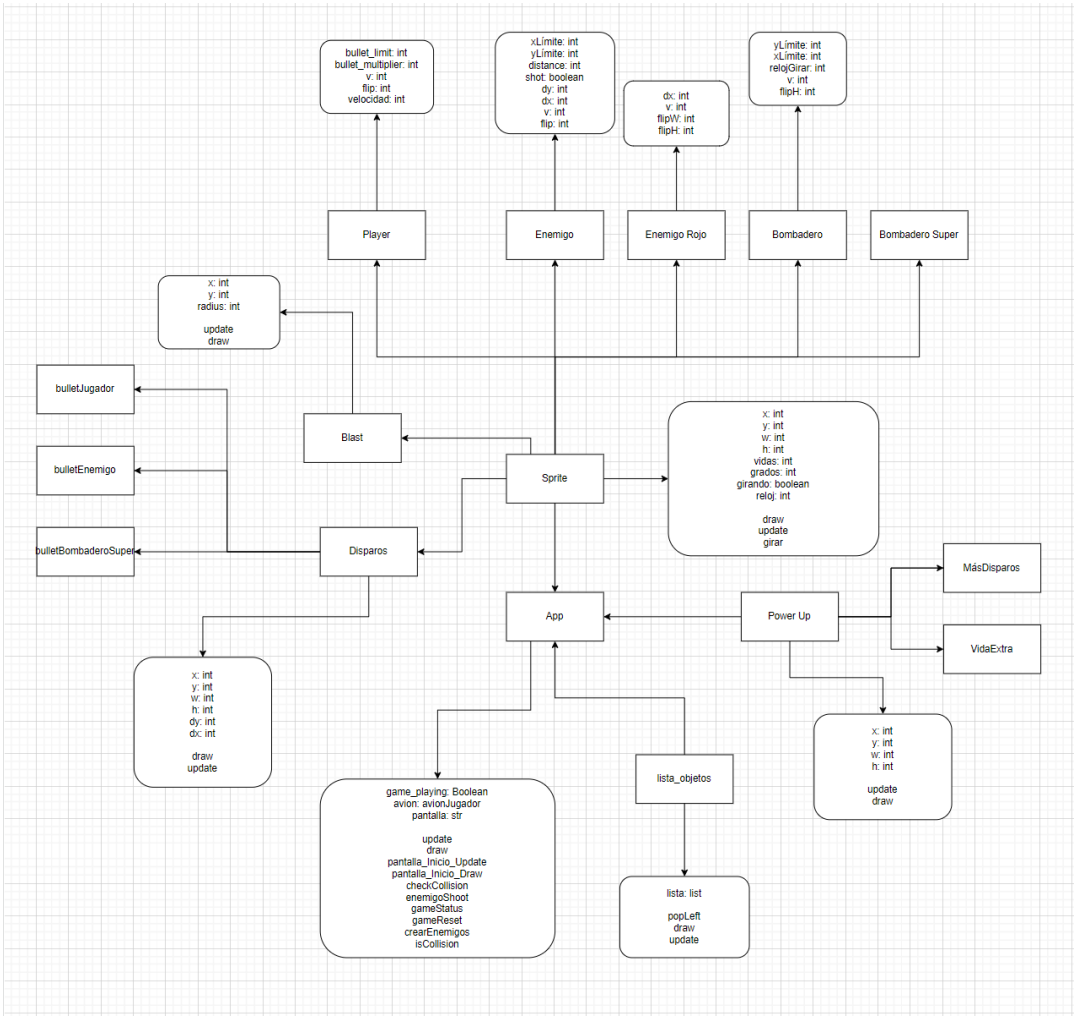
1942 es un juego en el que se pilota un avión de combate Lockheed-P-38 que necesita abatir a cuatro tipos de enemigos (Enemigo regular, rojo, bombardero, superbombardero) con tres vidas hasta alcanzar el máximo puntaje.

En el presente documento vamos a demostrar la metodología de trabajo con que se pudo lograr la construcción de este proyecto, auxiliado por el lenguaje de programación *Python* y el interpretador *Pyxel.*

Instrucciones

Para poder abrir 1942, es necesario que en el archivo ZIP del mismo nombre, se importe la carpeta de proyecto con todos los *assets* y *ficheros* incluidos a la carpeta *Pycharmprojects*, para posteriormente abrir la susodicha dentro del IDE de Pycharm.

Diseño de clases



Métodos y algoritmos más relevantes

Los enemigos, las explosiones y los disparos están en una lista de objetos (`class lista_objetos:`). Esta lista se itera por sí misma, llamando las funciones de update (`def update(self):`) y draw (`def draw(self):`) de cada objeto en la lista. De esta manera nosotros podemos tener seguimiento de todas las posiciones de los objetos en la pantalla.

```
class lista_objetos:

    def __init__(self) -> None:

        self.lista = []

    def popLeft(self):

        self.lista = self.lista[1:]

    def draw(self):

        for j in self.lista:

            j.draw()

    def update(self):

        for j in self.lista:

            j.update()
```

Cómo funcionan los disparos.

Así que hay una clase que es lista objetos que cuando se crea se inicializa a sí mismo e inicia UNA lista y al principio del programa hacemos una lista de enemigos, disparos y explosiones, además de disparos de enemigos. Así que hace una lista llamada lista de objetos, luego, la rellenamos con objetos de cada tipo, (disparo, enemigos y explosiones), por tanto es una lista que se llena de objetos diferentes.

Por ejemplo, llamamos shot_bullet.update la cual es una función de la clase de lista_objetos; luego la update de lista_objetos itera a través de la lista de shot_bullet, llamando la función update de cada objeto en la lista.

Posteriori la función update que se llama para cada objeto de la lista disminuye el valor “y” por 10 unidades y verifica si la bala ha salido de la pantalla, y si es verdad, lo elimina de la lista de objetos.

```
class bulletJugador:

    def __init__(self, x, y) -> None:

        self.x = x

        self.y = y

        self.w = 1

        self.h = 4

        shot_bullet.lista.append(self)

    def draw(self):

        pyxel.blt(self.x, self.y, 0, 48, 16, self.w, self.h, 0)

    def update(self):

        self.y -= 10

        if self.y < -2:

            shot_bullet.popLeft()
```

Iscollision

Para iscollision, requerimos de CUATRO parámetros x1, y2, x2, y2 y encuentra la distancia entre dos puntos usando la formula geométricade distancia, si la distancia es igual o menor a 8, collision va a regresar true, por lo contrario, false. Usamos esta función para la detección de colisiones en el juego.

Hay 4 tipos distintos de colisión entre los 4 tipos de objeto en el juego: jugador, enemigos, bullets de jugador, y bullets de enemy.

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Para `bulletEnemigo:`

Recibe 4 parametros, la posición del avión (`def __init__(self, x, y, avionX, avionY) -> None:`) enemigo donde está creado y la posición del avión jugador cuando esta siendo creado. Cuando se inicializa encuentra la distancia entre los dos puntos de coordenadas; luego la diferencia entre los valores de “y” de cada coordenada se dividen entre la distancia.

```
self.distance = (((self.x-self.avionX) ** 2) + ((self.y-self.avionY) ** 2)) ** 0.5

        self.dy = (self.avionY - self.y)/self.distance

        self.dx = (self.avionX - self.x)/self.distance
```

Lo anteriormente descrito; el algoritmo para definir self.distance y demás coordenadas.

Luego el mismo proceso se aplica para los valores de “x”. En términos matemáticos lo que hemos hecho es la normalización de un vector para que la bala se dirija hacia al avión jugador.

```
class bulletEnemigo:

    def __init__(self, x, y, avionX, avionY) -> None:

        self.x = x

        self.y = y

        self.w = 4

        self.h = 4

        self.avionX = avionX

        self.avionY = avionY

        self.distance = (((self.x-self.avionX) ** 2) + ((self.y-self.avionY) ** 2)) ** 0.5

        self.dy = (self.avionY - self.y)/self.distance

        self.dx = (self.avionX - self.x)/self.distance

        shot_bullet_enemigos.lista.append(self)

    def draw(self):

        pyxel.blt(self.x, self.y, 0, 51, 16, self.w, self.h, 0)
```

```
    def update(self):

        self.y += 2 * self.dy

        self.x += 2 * self.dx

        if self.y > HEIGHT:

            shot_bullet_enemigos.lista.remove(self)
```

Funcionalidades extras

Power-Ups

Más Disparos

Este power-up aumenta el límite de disparos que el avión puede disparar, y aumenta la cantidad de disparos que aparecen. El método avionShoot dentro de la clase avionJugador automáticamente coloca los disparos uniformemente a respeto de cuantos disparos hay cada vez que se pulsa el botón X. Solo puedes ganar este power-up tres veces.

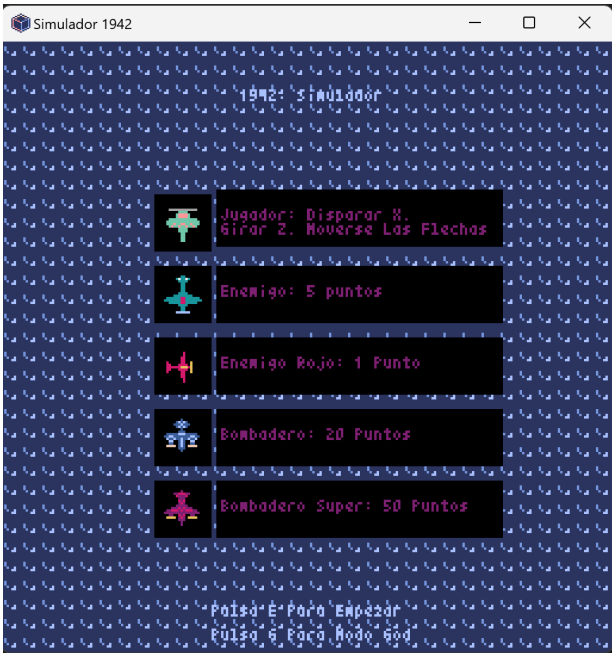
VidaExtra

Este power-up aumenta el número de vidas que tiene el avión del jugador. No hay límite de vidas que puede tener el jugador, pero este power-up no aparece muy frecuente. Con ganar muchas vidas extras puedes alcanzar una puntuación muy alta en el juego.

Modo “God”

Si se pulsa el botón ‘G’ antes de acceder el juego, se realiza el modo “god” en que el jugador no puede morir. Puede disparar constantemente e infinitamente, y moverse más rápido.

Pantalla de Inicio



La pantalla de inicio muestra el nombre del juego, las instrucciones para acceder el juego, y los aviones en el juego. Muestra cada tipo de enemigo, su nombre, y cuantos puntos vale. Muestra el avión del jugador y los botones para controlarlo. Z para girar, X para disparar, y las flechas para moverse.

Partes no Implementadas

Hemos creado un juego operativo, pero faltan algunas cosas de los Sprints. No hemos creado islas que aparecen en el fondo del juego, tampoco animamos las hélices de los enemigos. Todos nuestros atributos son públicos, y habría sido mejor si algunos fueran privados.

Conclusión

La práctica se ha completado usando los *sprints* dados por la docencia; se ha logrado la implementación de los requisitos impuestos en este trabajo para replicar el juego *1942*. Se han podido añadir elementos como *power-ups* además de sprites con aspectos refinados.

El principal obstáculo que tuvimos que afrontar fue el de cómo compartir el código y estar de acuerdo sobre distintas ideas a implementar en el programa. El IDE usado no tiene función para poder editar código con un compañero en tiempo real-análogo a la edición de documentos en *Google Docs*-por lo que se tuvo que constantemente mandar nuevas ediciones del código por correo luego de ponerse de acuerdo por aplicaciones de mensajería sobre qué cambios aplicar al programa.

Para finalizar, consta remarcar que el proyecto nos ha puesto a prueba todos los conocimientos adquiridos durante este curso de programación, y que además, muchos de los aspectos de *Pyxel*- interpretador de Python con que se realizó este juego- fueron aprendidos de manera autodidacta, por lo que hacer este proyecto fue un reto que nos ayudó a avanzar como programadores en este lenguaje, arduo trabajo que tuvo su recompensa en forma de mejor ética de trabajo y conocimiento en el área.

Problemas con función turning

Pyxel no permite voltear una imágen durante una animación debido a que es un motor de juegos *retro*, por lo que se tuvo que hacer código que hiciese esa función.

Básicamente lo que tuvimos fue una atributo que era self.grados que cambiaba, entonces para cada grado cambia, la “y” cambia, lo cual, asimismo cambia el self.b y determina cual imágen es dibujada en el programa con la función “draw”. Lo que sucede es que cuando algo necesita hacer el movimiento *turn* (voltearse) muchas veces, tendría que ejecutarse muchos statements *if-else*, algo que resulta ineficaz y poco práctico desde el punto de vista de programación.