MEMORIA PROYECTO FINAL: 1942

Creado por Álvaro Carrasco y
Pablo Amor

ÍNDICE:

RESUMEN DEL PROYECTO:

Con este proyecto hemos tratado de versionar el popular juego “1942”, añadiéndole algunos elementos y funcionalidades a nuestro gusto. El objetivo principal del juego es conseguir la mayor puntuación posible antes de perder todas las vidas, pues ha sido programado de tal forma que el juego no tiene fin, la partida dura tanto como el jugador sea capaz de alargarla.

El jugador podrá controlar un avión capaz de disparar y esquivar proyectiles, mientras le van apareciendo de forma aleatoria naves enemigas regulares, además de otros tipos de enemigos cada cierto intervalo de tiempo establecido. Todos los enemigos poseen patrones de movimiento y disparo únicos para cada tipo, obligando al jugador a plantear distintas estrategias para abatirlos.

Como imagen del fondo se irán generando islas y olas de forma aleatoria para hacer más vistosa la jugabilidad

CLASE AVIÓN:

En esta clase se definen todas las propiedades y métodos que tendrá el avión controlado por el jugador. Como parámetros se introducen la “x” y la “y” que tendrá el avión al principio de cada partida. Los atributos que tendrá nuestro avión serán su sprite, su posición, su velocidad, sus vidas y loops restantes, la cadencia de disparo, así como un atributo que guarda el instante en el que disparó por última vez, el tiempo de invulnerabilidad tras haber sido alcanzado por un proyectil y booleanos que indican si está haciendo la voltereta, si es invulnerable y si ha sido abatido.

Los métodos de esta clase son siguientes:

Move: Mueve el avión en función de la dirección que haya sido introducida como parámetro a la velocidad especificada en el atributo.

Helices: Recibe el número del frame en el que el jugador se encuentra de partida y según la paridad de este alterna entre dos sprites del avión que animan las hélices.

Voltereta: Recibe como parámetro el instante en el que comienza el loop del avión y según aumenta la diferencia de este con el frame actual el sprite del avión cambia intentando imitar una vuelta sobre sí mismo. Una vez terminada, cambia el atributo “loop” a False

Impacto: Recibe como parámetro el instante en el que el avión fue alcanzado por un proyectil enemigo y activa la invulnerabilidad del avión. Además, cambia el sprite a una explosión y después hace que el avión parpadee rápidamente hasta que deje de ser invulnerable.

Muerte: Recibe como parámetro el instante en el que el avión fue abatido y según la diferencia de este con el frame actual varía entre distintos sprites para animar una explosión del avión, acabando la partida.

CLASE FONDO:

En esta clase hemos almacenado todos los métodos utilizados para conformar el fondo del videojuego. Los parámetros que hemos utilizado han sido “x” e “y” para la posición inicial y un “tipo” para evaluar a qué elemento aplicábamos los métodos. Los atributos de la clase son “x”, “tipo” y “avance”, esta última se trata de un booleano que hace que se muevan los objetos del fondo o no.

Hemos creado un método (move) que para producir una sensación de movimiento hace que los objetos vayan moviéndose hacia abajo del tablero a una velocidad constante, haciendo así que parezca que el avión va poco a poco avanzando por el tablero.

Los elementos que utilizamos en el fondo son el portaaviones inicial, tres islas, las cuáles son Ibiza, Puerto Rico y Huete, Cuenca (nuestro pueblo en común), olas, la imagen inicial de 1942, y la imagen final de “Game Over”.

Además, en caso de que el avión llegue a cierta parte del tablero, se activa el avance y los objetos se empiezan a mover a una velocidad mayor hacia abajo con el fin de dar una mayor sensación de velocidad.

CLASE PROYECTILES

Esta clase almacena toda la información relacionada con los proyectiles en nuestro juego. Esta recibe como parámetros la posición inicial del proyectil y booleanos que indican si este pertenece al jugador, si es un proyectil “misil” y si es un proyectil “bonus”.

Los atributos del proyectil se corresponden con los parámetros recibidos, a excepción del daño, que dependerá de si el jugador se encuentra en el bonus y del sprite, que variará según el tipo de proyectil.

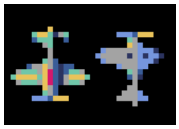
Esta clase cuenta con un único método move. Este define el movimiento que seguirá el proyectil. Si pertenece al jugador, se dirigirá a la parte superior de la pantalla a gran velocidad, mientras que, si es propio de un enemigo, irá hacia abajo a una velocidad menor. Hay una excepción en el caso del superbombardero, que disparará proyectiles más grandes, representados por un misil, cuyo patrón de movimiento se diferencia en que primero se posicionarán enfrente del avión del jugador (van teledirigidos) para luego caer en vertical sobre este, complicando en gran medida su esquive.

CLASE ENEMIGOS

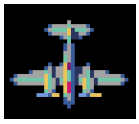
Esta clase almacena toda la información referente a los cuatro tipos de enemigos que existen en nuestro juego. Los parámetros de esta clase serán el tipo de enemigo que es, un timer para definir el movimiento en el caso de los enemigos rojos y un lado que se utilizará para ver el lado de spawn de los enemigos. Los atributos de esta clase serán su posición, su sprite, su vida, los puntos que da cuando se muere, la cadencia de disparo que tiene, el tipo y el timer.

Estas son las propiedades de cada enemigo:

a) **Regular:** El valor cadencia es muy alto y la probabilidad de que dispare normal, por ello solo suelen disparar una vez mientras están en el tablero. Mueren de un disparo y se generan de forma aleatoria. Otorga 100 puntos.



b) **Bombardero:** El valor cadencia no es alto y la probabilidad de que dispare tampoco, por ello dispara de forma bastante regular. Hacen falta varios disparos para abatirlo, se mueve a una velocidad lenta y aparece en intervalos de tiempo cortos. Otorga 500 puntos.



c) **Superbombardero:** Este dispara misiles teledirigidos, por lo que para que no dispare una gran cantidad de ellos hemos puesto que el valor de su cadencia sea normal pero la probabilidad sea baja. Hacen falta muchos disparos para abatirlo, se mueve muy lento, y aparece en intervalos de tiempo largos. Otorga 2000 puntos.

d) **Rojo:** La cadencia es alta y la probabilidad no lo es tanto, esto está hecho para que rara vez disparen. Mueren también de un disparo, aparecen de 5 en 5, se desplazan a una gran velocidad y aparecen en intervalos de tiempo medio. Otorga 150 puntos y en caso de eliminar a los 5 que salen otorga un bonus temporal que aumenta los proyectiles disparados por el jugador de 1 a 4.

El único método de la clase es el move, que mueve los enemigos dependiendo de que tipo sean: en el caso del regular, simplemente se desplaza hacia abajo moviéndose en diagonal hasta cierto punto y vuelve a subir; tanto el bombardero como el superbombardero, dependiendo del sitio por el que aparezcan, describen un rectángulo y se van por el lado contrario al que han aparecido, y los rojos o bien hacen un zigzag por la pantalla o describen tres triángulos mientras se deslazan en diagonal hasta acabar desapareciendo.

CLASE ENEMIGO MUERTO

Esta clase representa el resto que deja cada uno de los enemigos tras ser abatido, siendo este resto la cantidad de puntos que ha obtenido el jugador por la baja. Recibe como parámetros la posición en la que murió el enemigo y el tipo de este. Los atributos de esta clase son la posición que ocupa en pantalla el objeto, el tiempo que lleva este en pantalla y el sprite que toma cada uno de los objetos en función del tipo de enemigo al que hace referencia.

Esta clase solo posee un método “action” que desplaza hacia arriba el objeto y suma una unidad cada frame al atributo “tiempo” para que cuando este llegue a cierto valor se elimine el objeto.

CLASE TABLERO

Esta clase contiene toda la información del tablero en el que el usuario podrá disfrutar de cada partida a nuestro juego. Es la más amplia, pues en ella se encuentran todos los algoritmos que ejecutan los métodos que han sido definidos en el resto de las clases.

Estos son todos los atributos que incluye:

```
# Estos atributos definen las dimensiones del tablero
self.ancho = Constantes.ancho
self.alto = Constantes.alto

# En este otro grupo se definen diversos atributos de los que se
# harán uso a lo largo de la clase, así como los booleanos de inicio
# y final del juego y los marcadores de puntuación
self.inicio = True
self.gameOver = False
self.score = 0
self.high_score = 0
self.fondoAvance = False
self.timerGeneral = 0
self.timerVoltereta = 0
self.timerImpacto = 0
self.puntosBonus = 0
self.timerBonus = 0
self.disparoBonus = False
self.lado_rojo = 0
```

```
# En este grupo se introducen todos los atributos que corresponden a
# los elementos que se mostrarán en pantalla, introduciendo al
# inicio del juego el avión del jugador, el portaaviones, el título
# del juego (que hemos decidido que descienda con el portaaviones) y
# las olas iniciales.
self.avion = Avion(self.ancho // 2 - 11, 200)
self.lista_objetos = [Fondo(self.ancho // 2 -
                            Constantes.spritesFondo[0][3] // 2, 0,
                            "Portaaviones"),
                      Fondo(self.ancho // 2 -
                            Constantes.spritesFondo[3][3] // 2, 0,
                            "Titulo")]

self.lista_proyectiles = []
self.lista_enemigos = []
self.lista_olas = []
self.lista_enemigos_muertos = []
```

Como se puede ver los distintos atributos se pueden separar en estados del juego, puntuaciones, localizadores temporales para diversas acciones que transcurrirán a lo largo de varios frames y elementos que ocuparán un lugar en la pantalla como lo puede ser nuestro avión o las listas de enemigos y elementos del fondo.

Al principio de cada partida, ya en el método update se colocan en pantalla el avión, el portaaviones, el título y se generan grupos de olas instantáneamente en el fondo. Se inicia el timer propio de la partida para no alterar el spawn de los enemigos y se animan las hélices del avión.

Ahora se pasarán a explicar todos los algoritmos que realiza el programa una vez empezada la partida.

Movimiento de los objetos: Para el avión del jugador, se evalúan las pulsaciones en las flechas direccionales y si se está chocando con uno de los límites establecidos para poder avanzar. Si se choca con el límite superior, en la mitad de la pantalla, se activará el avance rápido del fondo para crear sensación de velocidad mayor. Para el resto de los objetos, se ejecuta un bucle for en su correspondiente lista llamando a cada respectivo método move.

Voltereta del avión: Se evalúa si se pulsa la tecla [X], y en caso afirmativo si el jugador tiene loops disponibles se cambia el booleano loop del avión y comienza la animación.

Spawns: Se generan, en base a un resto con el timer de la partida, todos los elementos dependientes de dicho timer, a excepción de los enemigos regulares, que se generan de forma aleatoria. En el caso de las olas, aparecen con un bucle for que genera varias de forma aleatoria a lo largo del ancho del tablero. Para los enemigos rojos, mediante varios “or” se evalúan desde un frame en concreto el resto en 5 saltos de 8 en 8 frames para poder hacer que salgan en formación, determinando el primero de ellos el modelo de movimiento a seguir guardándolo en el atributo self.lado_rojo.

Disparos: Para los disparos de cada objeto en pantalla se tendrá que cumplir que al darse la condición que le permita disparar la diferencia entre el momento de su último disparo y el frame actual deba ser mayor a una cadencia establecida para cada uno, también para el jugador. Este disparará cuando se pulse la tecla [Z] y no esté haciendo un loop. Para el resto de los enemigos, en lugar de presionar

un botón cada uno disparará de forma aleatoria, con una probabilidad determinada para cada tipo de enemigo. Los enemigos regulares solo podrán atacar cuando vayan hacia el jugador. Con estas condiciones, se agregará un proyectil a la lista de proyectiles desde la posición del atacante y con las propiedades que designe el mismo.

Colisiones: Para cada proyectil en la lista de proyectiles, en función de si el proyectil pertenece al jugador o no, se evaluará si está dentro de las dimensiones y la posición del sprite del jugador si es un proyectil enemigo, o de cada enemigo mediante un bucle for si es un proyectil disparado por el usuario. Si impacta sobre un enemigo, este perderá los puntos de vida que designe el daño del proyectil, y al llegar a 0, este generará un objeto en la lista de enemigos muertos, mostrando en pantalla los puntos que ha obtenido el jugador. Si se tratase de un enemigo rojo, se sumaría 1 punto al contador de puntos de estos y si se llegase a los establecido se activaría el bonus, cambiando el tipo de disparo. Además, después de cada enemigo rojo abatido se guarda el frame en el que ha ocurrido para una vez acabe la oleada se reseteen los puntos al cabo de un tiempo para conseguir que no se conserven de car a la siguiente. En caso de que sea el jugador el impactado, este perderá una vida y se activará el método impacto del avión. Al quedarse sin vidas, se ejecutará el método muerte del avión y se mostrará una animación de una explosión en el mismo, para posteriormente poner el atributo self.gameOver en True, acabando la partida y mostrando dicho mensaje en pantalla. Tras cualquier tipo de colisión el proyectil implicado se elimina de la lista de proyectiles.

Pasando al método draw, que representa los elementos indicados en pantalla, cabe destacar que cada uno de los objetos de las listas se muestran mediante un bucle for en su lista, con los parámetros que indican su posición en pantalla, su sprite como localizador en el archivo .pyxres que guarda los bancos de imágenes y con el negro como color transparente, pues es el fondo de dichos bancos de imágenes. Mientras se esté en partida, se dibujarán también textos con el número de vidas, de loops y las puntuaciones, permaneciendo el récord también incluso cuando termine el juego. En los momentos de inicio y final de partida, se dibujarán también un texto con el nuestros nombres y la instrucción para comenzar la misma, y una imagen con el letrero “GAME OVER” y las instrucciones para comenzar un nuevo intento, respectivamente. Previo a todo esto, es importante poner el comando pyxel.cls(color) para borrar todo lo del frame anterior y que no se vaya acumulando todo en pantalla, dicho color es un azul oscuro que representa el océano.

CONCLUSIONES:

Resumen del trabajo y partes adicionales y no implementadas:

El principal objetivo de este proyecto era versionar el juego “1942”, en el que un avión tiene que ir disparando a naves enemigas que intentarán abatirlo. Hemos implementado todo lo que se nos pedía en el guion, aunque algunas cosas como el movimiento de los aviones enemigos tienen patrones de movimiento que hemos ideado nosotros mismos. Nosotros hemos querido hacer un juego infinito, cuya principal finalidad es conseguir la mayor puntuación posible antes de que se te acaben las vidas. Además, en el caso del superbombardero hemos querido mostrar algo de originalidad y hemos hecho unos misiles teledirigidos. Algo que nos hubiera gustado implementar en el juego sería tanto una canción de fondo como sonidos al ser alcanzador por misiles y derrotar algunas naves, pero debido a no haber aprovechado bien el tiempo nos ha sido imposible.

Principales problemas encontrados:

Como principal inconveniente que se nos presentó fueron principalmente la inexperiencia a la hora de trabajar con la extensión “Pyxel”, de Python, pero con ayuda entre los compañeros y preguntando por los foros pudimos solventar la mayoría de dudas. Otro de los principales problemas fue el trabajar con el editor de Pyxel y los bancos de imágenes, que además de que no nos ofrecían espacio suficiente para los elementos que queríamos incorporar en pantalla, en numerosas ocasiones el editor no se nos ejecutaba en nuestros ordenadores, pero con el tiempo supimos arreglar los problemas y optimizamos el espacio todo lo que pudimos.

Valoraciones y aspectos a mejorar:

Hemos aprendido mucho acerca de lo realmente complicado que puede llegar a ser desarrollar un videojuego con uno tan simple como lo puede ser este, aunque lo cierto sea que es bastante entrenido. Como aspectos a mejorar de esta experiencia podemos destacar el no haber aprovechado bien el tiempo, principalmente por la falta de experiencia, con lo que nos deteníamos durante horas en el avance del juego por diversos problemas con el código, llegando a quebrarnos la cabeza hasta que encontrábamos una solución medianamente viable, si no llegábamos a descartar la idea. Tras haber terminado este proyecto, estamos seguros de que desarrollar otro videojuego parecido con la misma herramienta nos resultaría mucho más sencillo y placentero.