uc3m | Universidad Carlos III de Madrid

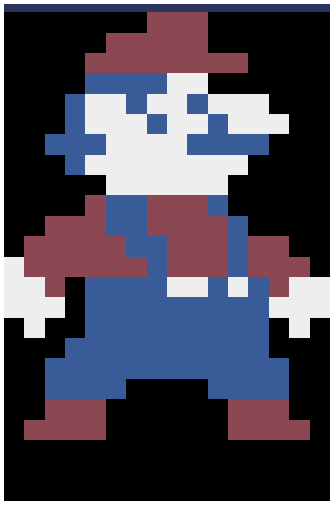
Grado Universitario en Ingeniería Informática
2023 - 2024

Programación grupo 84

“Mario Bros 1983”

Juan Francisco Salor Búrdalo

Alejandro Rodríguez González



Índice

Introducción.

Bienvenidos al juego de Mario Bros Retro. El objetivo del juego es pasar los diferentes niveles con las distintas hordas de enemigos, logrando que no se agoten tus vidas, del caso contrario habrás perdido y tendrás que volver a empezar, ¿lograrás pasarte todos los niveles o desarrollar un dolor de cabeza debido a la dificultad del juego?

Nada más empezar comprobaras que el fontanero más conocido de la historia de los videojuegos, Mario, se encuentra debajo de la pantalla y mediante los botones podrás desplazarte a lo largo del mapa. Para eliminar a los enemigos tendrás que golpear de abajo a arriba las plataformas cuando los enemigos estén pasando por encima de Mario. Si consigues golpear a nuestros malvados enemigos, enhorabuena, pero aún no habrás acabado con ellos. Para ello, tienes que golpearles lateralmente mientras que se encuentran tumbados, pero date prisa, que no van a durar tumbados toda la vida y se levantarán para buscar su venganza.

Parece fácil ¿verdad?, pues el juego está plagado de misterios como monedas extras que van apareciendo y tendrás que cogerlas para saciar la sed de oro que tiene Mario, o diferentes enemigos que irán apareciendo poco a poco, aumentando la dificultad, cuyo objetivo es acabar con nuestro fontanero.

¡Adéntrate en nuestro mundo, acaba con los malvados enemigos y consigue la máxima puntuación!

¿Cómo descargarlo?

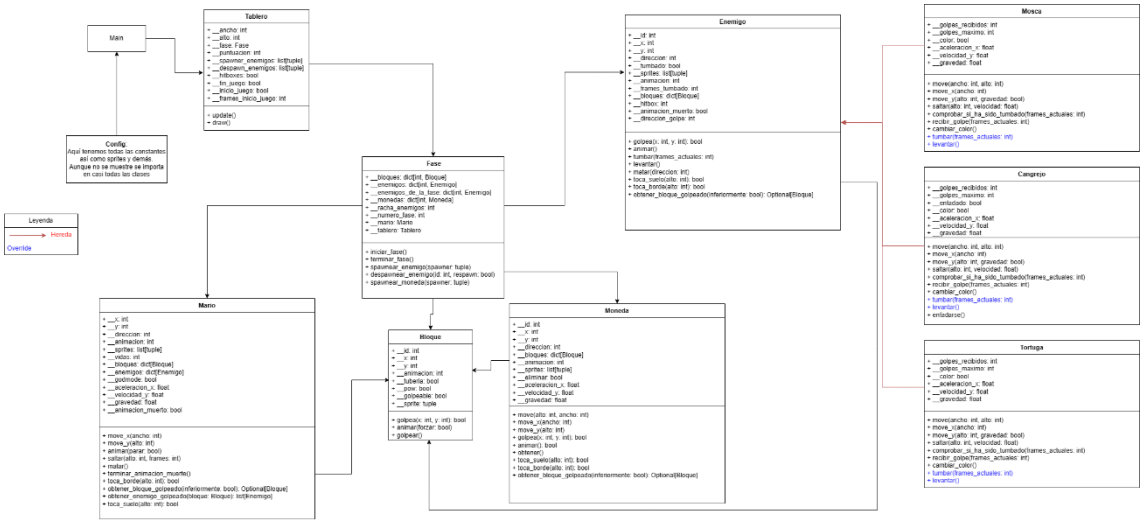
Este gran juego cuenta con una forma de descargarlo muy sencilla, pero ojo, hay que hacerlo de forma correcta y sin saltarse ningún paso.

Lo primero es descargarse el archivo .zip comprimido con todos los archivos, una vez descargado descomprimes el archivo, y procedes a instalar la librería pyxel, si es que aún no la tienes.

Para jugar debes ejecutar el archivo main.py, que será el encargado de poner en marcha todo el juego.

Además, no hay que preocuparse de las imágenes que usara el juego ya que se han incluido en la carpeta *assets*, es *Plug and Play*.

Diseño de clases



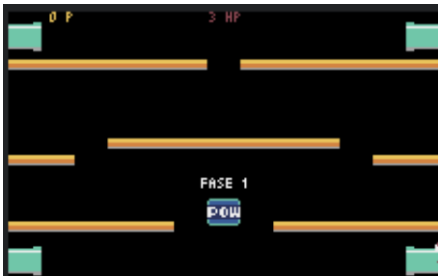
Cabe destacar que cada atributo en caso de que lo necesite tiene su correspondiente *getter y setter.*

Elementos del juego

La creación del mapa

Lo primero que tenemos que desarrollar para empezar a crear el juego, es el mapa. Nuestro mapa está creado por diferentes plataformas de 16 de largo, y con distintos bucles las hemos ido alargando según las necesidades, además de tuberías por donde saldrán los enemigos. Una vez hemos representado todo, tenemos que crear las físicas de cada uno de estos elementos, haciendo que si Mario se aparece en la parte inferior de la pantalla no se caiga debido a la gravedad o que cuando Mario suba a una de las bases de arriba no se caiga otra vez debajo de la pantalla. Para hacer que Mario no atraviese las plataformas, primero hay que comprobar si realmente toca una plataforma o no, después establecemos su "velocidad_y" en 0 para que deje de caer y además establecemos su y manualmente, imposibilitando así que se caiga.

Después de todo esto, hay que hacer que si Mario sale por la izquierda aparezca por la derecha y viceversa, esto se puede hacer con condicionales y cambiando el atributo "x" de Mario, de una forma muy sencilla.



Las *hitboxes*

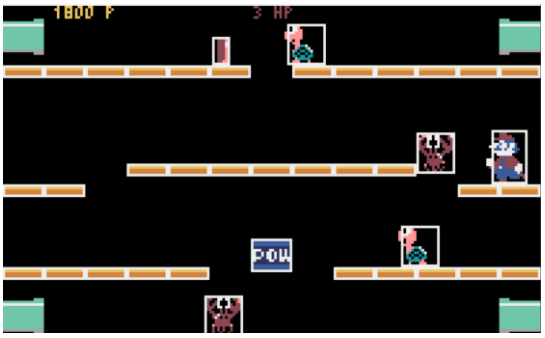
Para implementar las físicas, hemos utilizado un sistema de *hitboxes*, es decir, todo lo que puede colisionar, está envuelto de un "aura" invisible, de tal forma que si un punto está dentro de esa aura, significa que hay colisión. Estas han sido un elemento clave para la creación y detección de las colisiones. Sin las hitbox los personajes son simplemente imágenes en el mapa sin físicas y nada realistas. También nos permiten mantenernos en las plataformas como hemos mencionado antes y nos permiten detectar los choques entre Mario y los personajes, muy importante para restar vidas.

En nuestro juego puedes comprobar las hitbox pulsando la tecla 'E', nosotros las hemos creado rectangulares, ya que es mucho más sencillo manejarlas de esa forma.

La lógica que hemos usado para implementar las hitbox es la siguiente. Cogemos de referencia las coordenadas de la esquina superior izquierda de los personajes o figuras. A esta coordenada, sumamos el largo y el ancho propio de cada personaje para así cubrir casi toda la superficie del personaje y lograr que su hitbox le rodee a él por completo. Por lo tanto, alrededor del personaje figuran 4 puntos en cada uno de sus extremos que unidos entre sí recrean un rectángulo que configura la hitbox.

Aunque existe un pequeño problema, y es que los enemigos tienen ancho 16, y la separación que hay entre algunas plataformas es también de 16, lo que hacía que el enemigo o Mario, le costase mucho entrar. Para ello hemos tenido que realizar un pequeño ajuste de *hitbox*. En caso

de Mario, su *hitbox* ha variado 2 pixeles tanto a la izquierda como a la derecha, en caso de las moscas igual, y en el resto de los enemigos solo ha variado 1 píxel.



El movimiento

Para que Mario pueda moverse y podamos controlarle, hemos añadido varias líneas de código. El problema es cuando tenemos que añadir la gravedad y el movimiento autónomo de los diferentes enemigos.

Lo primero ha sido crear la gravedad, función básica en casi cualquier videojuego. Nosotros decidimos tirar de ecuaciones físicas y generar una especie de movimiento rectilíneo uniformemente acelerado. Si controlando a Mario le das al botón de saltar y se encuentra tocando suelo, se le otorgara una “velocidad_y” negativa, que se sumará a su coordenada y constantemente, pero para que se vaya rebajando esta como si saltásemos en la vida real lo que haremos es ir sumando a la “velocidad_y” una constante llamada gravedad por lo tanto se irá desplazando cada vez más lento. Cuando la “velocidad_y” llegue a 0, se le seguirá sumando la gravedad y entonces se desplazará en sentido positivo del eje, simulando la caída, y cuando toque la plataforma por las físicas de esta, su “velocidad_y” se restablece a 0 quedando estático. Para los enemigos, cuando estos no se encuentren en una plataforma (están en caída libre), se le irá sumando a esa velocidad_y inicial que es 0 la gravedad, simulando así la caída libre hasta que toque una plataforma de abajo y su velocidad_y se restablezca a 0.

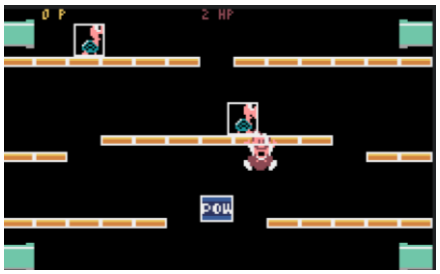
Los enemigos al ser totalmente autónomos se irán desplazando horizontalmente sin detenerse, a no ser que interactuemos con ellos, sufriendo el efecto de la gravedad. El movimiento de las moscas es algo distinto, ya que además de moverse solo horizontalmente y ser afectadas por la gravedad, saltan, algo que ocurre con Mario, y para hacerlo de forma autónoma hacemos que cada vez que se mueva y toque suelo, salte, además de moverse horizontalmente.

Las interacciones

La base para que los elementos interaccionen entre ellos se basa en la hitbox de cada uno. Tendremos varias interacciones, Mario contra los enemigos de forma horizontal y Mario contra las plataformas de forma vertical, además de Mario recogiendo monedas.

La interacción lateral de Mario contra los enemigos es muy sencilla. Si la hitbox de Mario y la del enemigo coinciden y el enemigo esta tumbado, Mario ha derrotado a un enemigo y se ejecuta la animación. Sin embargo, si el enemigo está de pie, a Mario se le restara una vida y se ejecuta su animación correspondiente.

La interacción de Mario con la plataforma es también muy sencilla. Si la parte superior de la hitbox de Mario coincide con la plataforma se cambiará el sprite de la plataforma, y si el sprite de la plataforma que se ha modificado coincide con la hitbox de un enemigo le tumbaremos.

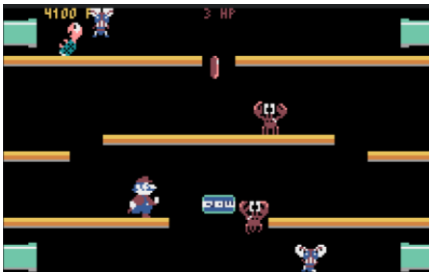


Las fases

Los malvados enemigos no dejaran salirse con la suya a Mario tan fácilmente. Es por eso por lo que a medida que vas subiendo de ronda aparecerán más enemigos cada vez más poderosos y numerosos para hacer frente a nuestro protagonista.

Hemos utilizado un algoritmo de generación de enemigos propia y totalmente personalizable desde el archivo “config.py”

Hemos creado una constante que será el numero mínimo de enemigos, que recibe el valor de 4 por defecto. La cantidad de enemigos será el numero de esa constante multiplicado por el número de ronda. Por ejemplo, en la ronda 2 aparecerán ocho enemigos, en la ronda 3 doce enemigos y así sucesivamente. Hemos seguido un orden de dificultad progresiva, apareciendo en la ronda 1 las tortugas, la ronda 2 los cangrejos y en la ronda 3 las mariposas. Sin embargo, como nos parecía muy fácil pasarse el juego, decidimos hacerlo como cualquier juego arcade, que tenga infinitos niveles. Cada uno de los niveles será cada vez más difícil que el anterior y cuyo objetivo es tener la mayor puntuación jamás lograda.



Pantallas de inicio y fin

Al terminar el juego de forma funcional viendo que cumplía todos los requisitos del proyecto decidimos pasar un pincel a nuestro proyecto y añadir cosas que le diera un toque más atractivo. Para ello hemos añadido una pantalla de inicio, donde te de la opción de iniciar el juego y te explica los controles. También hemos añadido una pantalla de fin, donde te muestra el récord de puntos obtenidos.

Lo primero que vemos al ejecutar el juego es la pantalla de inicio. Cuando damos a comenzar el juego, un booleano cambiara a True y nos permite ejecutar el juego. Cuando perdemos, otro booleano se cambia a True y el juego finaliza. Esta lógica de usar los booleanos como una forma de mostrar una pantalla u otra nos abre un mundo de posibilidades, ya que si en un futuro

quisiéramos seguir modificando el juego con esta lógica nos sería fácil cambiar de un modo de juego e incluso de un juego a otro. Tan solo tendríamos que mostrar el booleano que queremos cambiar y por lo tanto qué modo de juego o mapa quisiéramos ejecutar.

Pantalla de inicio:

Pantalla de fin:

Elementos de programación usados

Para la creación del juego hemos tenido que implementar una gran cantidad de métodos y lógicas de la programación.

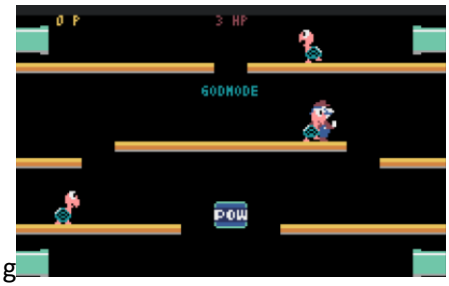
Hemos hecho un gran uso de los métodos de programación básicos como los condicionales para dar pie a la ejecución o no de algunas partes del código o la implementación de listas y diccionarios para el almacenamiento de datos, así como los distintos objetos de enemigos, bloques, etc.

La joya de la corona en este proyecto ha sido la POO. El uso de clases, atributos y métodos propios de una clase determinada ha sido el verdadero reto de este proyecto, ya que hemos descubierto una forma mucho más compleja de programar, pero mucho más útil y ordenada. Hemos dado un gran uso al encapsulamiento, a la herencia de los atributos de una clase a otra consiguiendo un código más sofisticado y fácil de leer y de entender por otra persona.

God Mode

La depuración de errores también conocido como *debugging*, es lo más común a la hora de programar. Por este motivo hemos decidido implementar un modo invencible de Mario para probar cada una de las mecánicas del juego de formas más cómoda sin tener el inconveniente de sobrevivir un número definido de rondas o evitar que te maten los enemigos.

Esto lo implementamos desde el principio, ya que a nuestro parecer junto el “print()”, es una de las herramientas más útiles para depurar errores. Este modo se activa pulsando la letra “G”.



g

Conclusión

Hemos logrado resolver el problema de la creación de un videojuego de Mario Bros estilo retro consiguiendo que funcione de forma correcta y que se pueda jugar a la perfección.

Debido al tiempo limitado de la practica no nos ha dado a implementar más cosas extras, como los distintos enemigos o incluso estructurar más aún el código, aunque es bastante escalable, nos hubiese gustado crear algún par más de clases como una clase “Pantalla” para crear distintas pantallas, una clase “*Hitbox*” para controlarlas de forma más estructurada e incluso una clase “Movimiento” para evitar repetir mucho código.

Nos gustaría acabar con un mensaje gratificante, ya que en el desarrollo de este juego hemos tenido momentos de estrés y de impotencia debido al desconocimiento de como hacer una serie de cosas. Sin embargo, hemos aprendido a resolver los problemas y hemos aprendido un nuevo enfoque de la programación, motivándonos a seguramente en un futuro desarrollar otro juego o quizás seguir desarrollando este.