SUPER MARIO BROS

POR

CARLOS SEGUÍ

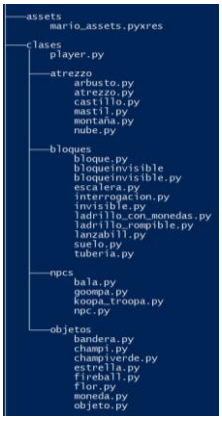
Y

RAUL AGUILAR

MEMORIA

Tabla de contenido

Diseño de clases



Hemos dividido cada clase en una carpeta, y cada subclase en un archivo dentro de dicha carpeta.

Las clases se dividen en 6 superclases:

- game
- atrezzo
- bloque
- objeto
- npc
- player

Game

Es la clase principal, así mismo es el único archivo que debe ser ejecutado. La clase game cuenta con 17 métodos, cada uno con una función. En un principio íbamos a crear una clase menú y una clase nivel, sin embargo decidimos integrar ambas en la misma clase ya que generaba mucha complejidad y no ayudaba a la legibilidad del codigo, además suponía cambiar los cimientos de lo que ya estaba desarrollado.

Update

El método tiene tres preguntas para distinguir si nos encontramos en el menú de inicio, en el de muerte o en el propio nivel. Si nos encontramos en el menú de muerte y pulsamos intro llamara al método reset_level(), a no ser que hayamos perdido todas las vidas en cuyo caso llamara al método reset_game(). Cuando se realiza la ejecución del nivel llama a métodos propios de cada clase que actualizan el estado de todos los objetos. Además ejecuta el método mantener_jugador_en_pantalla() y borrar_entidades()

Draw

Se encarga de dibujar todo en pantalla. Sigue las mismas tres preguntas que update para dibujar los menús o el nivel. Cuando dibuja el nivel, por optimización, solo dibuja los elementos que hay en el visor, para ello utiliza la pregunta

if not self.item_a_dibujar[i].coord[0]>1.5*pyxel.width:

para determinar si el elemento esta dentro o cerca del visor y dibujarlo y después va dibujando elemento a elemento mediante bucles for. Tambien tiene un apartado especial para dibujar al jugador y a los efectos de la estrella si es que tuviera. Por ultimo dibuja la interfaz.

Generar_suelo, objetos, atrezzo, bloques y npcs

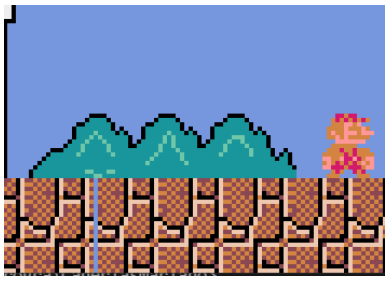
Son 5 metodos distintos encargados de crear una lista con las posiciones de los elementos del nivel. A pesar de que el suelo son bloques lo generamos aparte ya que usamos bucles while para generarlo mas cómodamente y pensamos que seria mas visual tenerlos por separado

Mantener jugador en pantalla y desplazar nivel

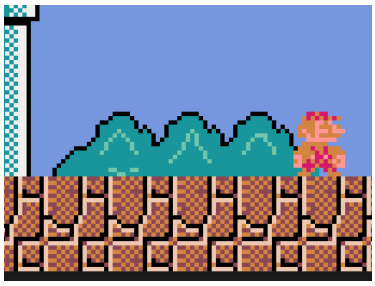
Detecta las coordenadas del jugador para que en caso de llegue a la mitad de la pantalla llamara al método desplazar_nivel() para transmitir su movimiento a los demás objetos (modificando sus coordenadas) haciendo así que el nivel se mueva a su alrededor.

Redondear

Al trabajar con velocidades decimales obtenemos una mejor sensación de juego, como por ejemplo con las inercias, la suavidad del salto, etc. Sin embargo esto significa que existen las coordenadas decimales y al dibujarse se aproximan a enteros, el método round de Python presenta el problema de que no siempre aproxima igual, ej: round(1.5)=2 y round(2.5)=2 cuando desearíamos un resultado más consistente que redondeara siempre hacia abajo (o arriba) a partir del 0.5, por lo tanto creamos este método que haciendo uso de round y restando un número muy pequeño permite aproximar los valores de forma consistente evitando que se formen huecos que a pesar de ser únicamente visuales no son nada estéticos.



Sin método redondear.



Con método redondear.

Reset level y game

Estos métodos reinician respectivamente el nivel y el juego, al reiniciar el nivel el jugador conserva únicamente sus vidas y al reiniciar el juego no se conserva nada, creamos un método reiniciar juego en vez de volver a llamar al __init__ de game ya que esto hacia que se reiniciara pyxel y se crearan mas ventanas, dando muchos problemas.

Borrar entidades

Se encarga de borrar todos los elementos que desaparecen (bloques, objetos, npcs muertos, etc), así como de borrar aquellos que abandonan el visor por la izquierda, los bloques que salen por la izquierda no son eliminados ya que si nos desplazamos rápido al eliminar tantos objetos generaba lag.

Attrezzo

Son los elementos de fondo como montañas, nubes y arbustos. También se han incluido aquellos objetos que no tienen ninguna estructura más que posición y sprite así como el mástil y el castillo . No tienen ningún tipo de colisión ni comportamiento, solo tienen estructura dividida en dos parámetros, su sprite y sus coordenadas.

Bloques

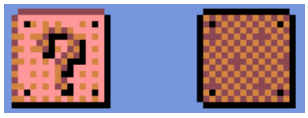
Consta de 7 subclases; escalera, interrogación, ladrillo con monedas, ladrillo rompible, suelo y tubería. Todos los bloques tienen en común los atributos de coordenadas, coordenadas iniciales(se usa para animaciones), sprite, ancho, alto, velocidad en y y existe (un booleano que sirve como marca para los bloques que deben ser eliminados). Todos los bloques cuentan con el método reposicionar, que les permite desplazarse un máximo de dos pixeles en el eje y a partir de las coordenadas iniciales.

Escalera y tubería

Se comportan igual, y lo único que tienen de especial es que tienen una altura variable que se debe introducir mediante un parámetro.

En la imagen ambas son la misma clase tubería pero con alturas diferentes.

Interrogación



Tiene un parámetro que indica si tiene monedas u objetos, si tiene monedas otorga un número al azar de ellas entre 1 y 6, además tiene unos frames de invulnerabilidad después de haber sido golpeado para no poder sacar todas las monedas de un solo salto, cuando se consumen todas las monedas cambia su sprite al de la derecha. Si tiene objetos otorgara una seta si lo golpea Mario o una flor de fuego si lo golpea Super Mario.

Ladrillo con monedas y rompible

El ladrillo con monedas tiene n monedas entre 1 y 6 y cuando se consumen todas desaparece, el ladrillo rompible desaparece al contacto sin otorgar nada o puede contener una estrella, en cuyo caso al contacto cambia su sprite al de un bloque de interrogación golpeado. Podríamos haberlos hecho con un solo bloque, sin embargo debían tener métodos distintos de golpear en función de su contenido, lo cual daba más problemas que soluciones.

Suelo

Es un bloque normal sin nada en especial que tiene la textura del suelo.

Lanzacohetes

El lanzacohetes tiene un parámetro que especifica se es alto o bajo, así como otro parámetro que indica en qué sentido dispara (izquierda derecha o ambos), el juego es el encargado de llamar a el método disparar de lanzacohetes, el cual añade un elemento bala a la lista de npcs

Bloque invisible

Se comporta como una interrogación pero con un sprite invisible, además solo puede otorgar un 1up

Objeto

Son aquellos objetos que tienen un movimiento y colisiones propias. Además influyen directamente en el estado del jugador.

Chami, flor y seta verde

Champi tiene un movimiento que consiste en saltar una vez cuando aparece y luego deslizarse, al colisionar lateralmente con escaleras y tuberías (los únicos bloques que aparecen al nivel del suelo) rebota, esta distinción se hace ya que si no golpea en bucle al colisionar con el suelo. Flor no tiene mas movimiento que un pequeño salto al aparecer. Ambos bloques interactúan con el jugador, cambiando su estado de Mario->Super Mario->Mario Fuego.

La seta verde tiene el mismo comportamiento que el champi salvo que otorga una vida extra

Fireball y estrella

Su movimiento lateral es igual al champi (rebotan con tuberías y escaleras) y su movimiento en el eje y consiste en rebotar y permanecer saltando todo el rato, la fireball desaparece al 3 rebote o al colisionar con un enemigo. Hemos modificado el sprite de Fireball para hacerlo mas grande ya que esto nos facilitaba las colisiones. La estrella desaparece en contacto con el jugador dándole 30 segundos de invulnerabilidad, matando así a todo lo que choque con él.

Bandera

Cuando colisiona con el jugador finaliza el nivel y hace que el jugador inicie la animación final.

Moneda

Únicamente salta hacia arriba al aparecer y desaparece en medio segundo, cuando desaparece suma uno al dinero del jugador.

Npc

Los npcs comparten los atributos de velocidad en x e y, un ancho y alto (para las colisiones), un bool es caparazón para las colisiones también (aunque solo los koopa troopas se vuelven caparazón el parámetro necesita estar en todos los npcs para poder verificarlo de forma genérica en todas las colisiones) y un está vivo que marca si deben ser eliminados. Tambien comparten los métodos sufrir gravedad, que hace que caigan, colisionar bloques, que hace que no caigan o que reboten al igual que el champi. Otro de colisionar npcs que hace que si colisionan entre si reboten a no ser que uno de los dos sea un caparazón en cuyo caso muere el otro. También pueden colisionar con fireballs que los remata y suma puntuación al jugador.

Todos los npcs tienen una comprobación en el método actualizar estado para que si están fuera del visor no se actualicen.

Koopa tropa

Solo puede morir al ser golpeado por un caparazón, una bola de fuego o cayéndose por un precipicio. Al ser golpeados por el jugador se convierten en un caparazón quieto, y al golpearlos otra vez se comienzan a mover. Al convertirse en caparazón, cambia su alto su sprite y su velocidad máxima, al cabo de 5 segundos vuelve a transformarse en un koopa normal

Goompa

Se escribe goomba pero lo pusimos mal y nos hizo gracia. Su movimiento es igual al del koopa cuando no es un caparazón y muere al ser golpeado superiormente por el jugador.

Bala

No les afecta la gravedad, tampoco el fuego y cumplen el parámetro es caparazón así que matan a otros npcs. También se destruye en contacto con otros bloques o cuando el jugador salta encima.

Player

Es la clase más importante y que más trabajo hace del juego. Los métodos más importantes son:

Colisionar bloques primero comprueba las colisiones en el eje y para hacer que no te caigas de los bloques y que los golpes al colisionar desde abajo, después comprueba las colisiones en el eje x para hacer que no atravieses bloques horizontalmente. Al colisionar por encima de un bloque automáticamente cambia tus coordenadas a las del bloque + tu alto para que no te quedes atravesándolo parcialmente si vas muy rápido, esto también causa que subas escalones automáticamente. también al golpear un bloque inferiormente se te aplica el doble de gravedad para evitar que no lo atravieses nunca.

Colisionar npcs detecta si chocas con un npc y si lo haces verticalmente u horizontalmente, si chocas verticalmente llamas al método colisionar_jugador de el npc y si chocas horizontalmente llamas a tu método recibir_daño. Además si eres estrella solo se considera si chocas de cualquier manera con el npc y llama a su método morir directamente. Al colisionar verticalmente te da 15 frames de invulnerabilidad para que no se reciba mas de un impacto seguido

Colisionar objetos simplemente detecta si colisionas un objeto y comprueba de que tipo es para cambiar el estado del jugador como sea necesario.

El método colisionar() en general es un método auxiliar que se usa en todos los anteriores y comprueba las coordenadas de una entidad respecto al jugador.

El método detectar botones detecta las teclas y le transmite velocidad al jugador, también detecta cuando no se pulsa nada para aplicar el coeficiente de rozamiento al jugador(esto hace que frene de una manera muy natural y no en seco).

Los métodos actualizar_animaciones. Coger_bandera, Fase1_bandera y fase2_bandera son métodos de animaciones que únicamente cambian el sprite del jugador para que se adapte a si esta andando saltando o deslizándose por la bandera. Funcionan mediante basicamente

if pyxel.frame_count % (c.fps/un_numero) == 0: cambiar sprite

que hace que se vayan alternando los sprites cada n fotogramas, tambien se detecta si la velocidad del jugador y hacia donde mira no coincide para derrapar y si estas saltando para poner el sprite de salto.

Lanzar fuego simplemente añade a la lista de objetos una bola de fuego en la dirección a la que estes mirando.

Por último el método reset_state reinicia los atributos que deben reiniciarse al comienzo del nivel y al morir.

Trabajo realizado

Trabajo básico

Hemos implementado el nivel 1 del juego base prácticamente entero, hay un movimiento del jugador satisfactorio, incluyendo animaciones de andar, saltar girar y agacharse de Mario pequeño, grande y de fuego, colisiones e interacción con el entorno, una interfaz grafica con el score de Mario, las monedas y el tiempo restante, hemos puesto decoración como montañas, arbustos y un generador aleatorio de nubes. Además hemos añadido todos los enemigos del nivel original, goombas y koopas, así como su interacción con el jugador y el entorno.

Los bloques reaccionan al interactuar con Mario y viceversa, la generación de objetos en bloques de interrogación es la apropiada así como el cambio de aspecto de estos, las monedas aumentan la puntuación de Mario, y cuando Mario mata enemigos también obtiene puntuación.

Mario puede obtener objetos especiales, la seta le hace grande, la flor de fuego le permite lanzar bolas y la estrella le hace imparable, todos estos objetos también otorgan puntuación.

Se tiene en cuenta el tiempo del nivel y al llegar a 0 se reinicia el nivel y Mario pierde una vida

Trabajo extra

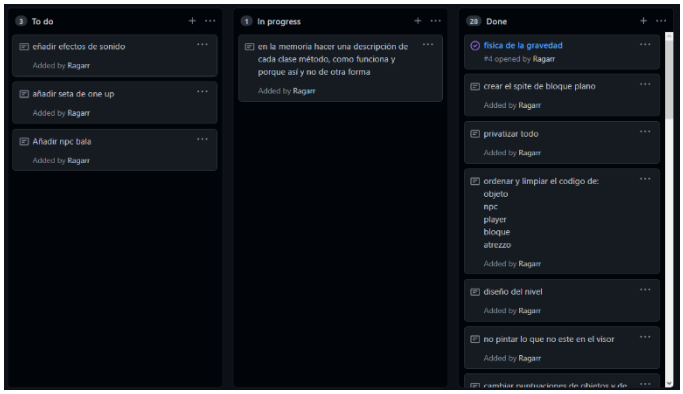
Se ha incluido la flor de fuego como un powerup extra que permite a Mario recibir un impacto más y disparar fuego que mata a los enemigos, también se ha incluido la estrella, la cual sale de un bloque de ladrillo aparente normal, y hace que Mario tenga una animación como de brillo y mate a todos los enemigos al tocarlos.

También hemos incluido una animación al inicio del juego, la bandera al final y las animaciones de andar girar saltar etc. Así como un menú de inicio y un menú de muerte con un recuento de las vidas restantes.

Organización y conclusiones

Organización del trabajo

Antes de comenzar el proyecto decidimos que para trabajar colaborativamente usaríamos un repositorio de GitHub. En un principio nos costó aprender a usar los repositorios sin embargo una vez entendido lo más básico descubrimos que son muy convenientes y útiles



En el aprovechamos la función de proyectos de GitHub para crear unos diagramas de trabajo que nos ayudaron a mejorar la organización y distribuirnos el trabajo correctamente, además al usar GitHub existe la ventaja de que sabemos que cambios ha hecho la otra persona para entender mejor su codigo.

Además el control de cambios nos ayudo mucho a corregir errores y localizar cambios erróneos.

También la posibilidad de trabajar con varias ramas nos permitía hacer trabajo paralelo y testear cosas sin afectar al código que ya era funcional.

Mayores problemas encontrados

Colisiones

El principal problema que hemos encontrado en el desarrollo del juego fue hacer un sistema de colisiones coherente, eficiente y universal. Cosa que al final hemos logrado en cierta medida, a pesar de que sigue habiendo algunas comprobaciones que deberían evitarse pero no hemos sido capaces de mejorar. Nos inspiramos en otros sistemas de colisiones AABB pero no fuimos capaces de crear un sistema AABB perfecto y tenemos una mezcla de AABB con algo propio nuestro.

Animaciones

La interacción de las animaciones con las colisiones nos ha dado bastantes problemas en general, sobre todo en las setas y demás objetos que aparecen dentro de bloques

Menú

```
Class game():
    ...
Class menú(game):
    ...
Class nivel (game):
    ...
```

En un principio planteamos los menús como una clase aparte de la siguiente forma, sin embargo, nos presentó tantos problemas a un nivel tan básico del juego que había que cambiar el funcionamiento de prácticamente todo lo que ya estaba hecho, así que decidimos apañarlo todo dentro de la misma clase game.

Critica al proyecto final

Colisiones

No nos encontramos nada satisfechos con cómo se comportan las colisiones internamente, dan el pego y apenas fallan de cara al usuario, sin embargo el sistema que usamos para comprobarlas no es universal para todos los bloques y en algunos casos da demasiadas vueltas para comprobar las cosas, sin embargo no teníamos más tiempo ni ideas para arreglarlo. Además a lo largo del desarrollo fuimos poniendo parches sobre parches en el sistema de colisiones, cosa que solucionamos y reeditamos lo mejor posible al final pero que durante el desarrollo nos causó muchísimos problemas.

La bandera y la animación final

El hecho de que la bandera sea un objeto separado en dos partes y que el mástil no haga absolutamente nada no nos convence en absoluto y refleja problemas en el diseño de clases, se podría arreglar creando unas clases con métodos dedicados exclusivamente para las animaciones, sin embargo al igual que en las colisiones no hemos tenido tiempo para implementar esto ya que suponía cambiar los cimientos de muchos comportamientos en el juego.