Práctica 3: Mario en Quintus

Guillermo Jiménez Díaz, Pedro A. González Calero

Entrega: 19 de abril de 2015, 23.55h

Introducción

En esta práctica vamos a desarrollar otro videojuego completo clásico utilizando el motor Quintus. El código fuente completo del que partir se entrega con la práctica aunque también se puede descargar desde su repositorio en Github.

Super Mario Bros

Super Mario Bros es el archiconocido videojuego de género arcade, publicado por Sega en 1981. Aunque el objetivo del juego original era muy ambicioso (sobrevivir a las hordas de Bowser y salvar a la princesa Toadstool) en esta práctica vamos a implementar algunas de las mecánicas principales del juego original. Junto con este enunciado se aporta un zip con varios ficheros que deberemos utilizar a lo largo del desarrollo, incluyendo una versión de Quintus estable.

Se recomienda que se implementen las mecánicas en el orden en el que se describen en este documento. De esta forma se garantiza un desarrollo incremental del juego sin miedo a quedarnos bloqueados en mecánicas que no podamos resolver.

Desarrollo de la práctica

Para la realización de esta práctica es *imprescindible* el uso de un servidor web ya que desde nuestro juego vamos a cargar archivos no imágenes y el protocolo `file://` nos denegará la carga de estos archivos.

Se recomienda empezar revisando la documentación de Quintus así como los ejemplos que vienen con el framework y que podéis encontrar por Internet. También se recomienda que se implementen las mecánicas en el orden en el que se describen en este documento. De esta forma se garantiza un desarrollo incremental del juego sin miedo a quedarnos bloqueados en mecánicas que no podamos resolver.

Carga del objeto Quintus

Crea un objeto Quintus para representar la pantalla principal del juego (320x480). Configúralo inicialmente con los módulos `Scenes`, `Input`, `UI` y `Touch`. No olvides añadir los archivos `.js` de cada uno de los módulos al HTML. Activa los controles por defecto de teclado y los eventos de touch (para los mensajes que aparecen en la ventana).

Carga del nivel TMX

Carga el nivel TMX que os proporcionamos como material adicional. Para cargarlo deberás usar los módulos `TMX` y `Anim`. Primero carga los assets con el método `loadTMX`. Luego crea una escena (llamada `level1`, por ejemplo) y carga en ella el mapa creado usando el método `stageTMX` que proporciona el módulo `TMX`.

Añade el módulo `2D` para hacer uso del componente `viewport`. Añádelo a la escena creada y centra la escena en la posición (150, 380). Deberías ver lo que aparece en la siguiente imagen.

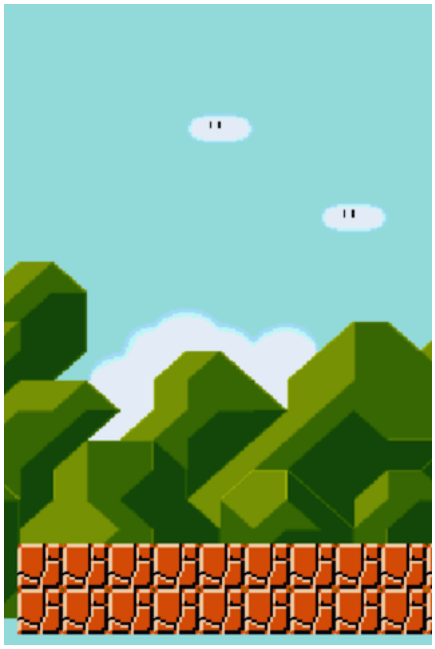

Figure 1: Escenario cargado

Mario en escena

Crea un `Sprite` llamado Mario para representar al jugador (necesitarás añadir, si no lo has hecho antes, el módulo `Sprite`). Carga y compila la hoja de sprites llamada `mario_small` y usa alguno de los sprites de la misma. Añádele los componentes `2d` y `platformerControls`. Añádelo a la escena (de nuevo, en la posición (150, 380)), haz que el viewport de la escena lo siga y ajústalo para que pueda pasar los obstáculos y que quede el viewport como en la imagen puesta a continuación (pista: juega con el offset del componente `viewport` de la escena).

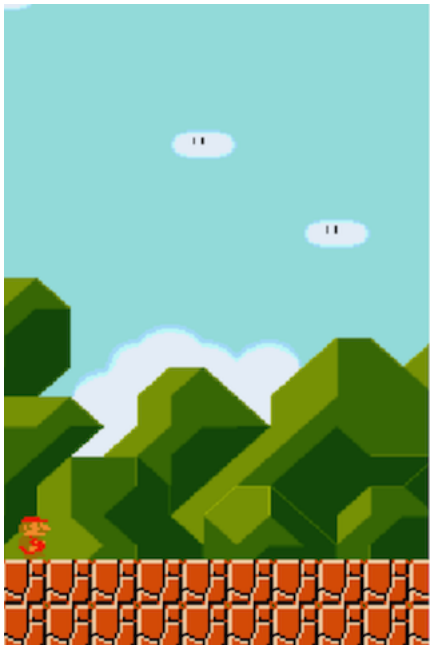

Figure 2: Mario aparece en escena

En el método step de Mario haz que éste vuelva a la posición inicial en el caso de que se caiga del escenario. Puedes considerar que este método es lo que hace Mario cuando muere.

Goomba

Crea un `Sprite` llamado Goomba para representar al primero de los enemigos de Mario. Carga y compila la hoja de sprites llamada `goomba` y usa alguno de los sprites contenidos en ella. Añade el componente `2d` y utiliza los eventos de `bump` de colisión definidos por este componente para hacer que Mario muera si colisiona lateralmente con Goomba pero que muera Goomba si Mario salta encima de él. Añádelo a la escena.

Una vez que tiene este comportamiento añádele el componente `aiBounce` para que se mueva en una dirección y rebote en caso de chocar contra algo (que no sea el jugador, claro). Colócalo en una zona del escenario en la que pueda colisionar lateralmente con dos paredes y añádele una velocidad horizontal inicial.

Bloopa

Crea un `Sprite` llamado Bloopa para representar otro enemigo de Mario. Usa alguno de los sprites de la hoja de sprites llamada `bloopa`. Añade el componente `2d` y utiliza los eventos `bump` de colisión definidos por este componente para hacer que Mario muera si colisiona lateralmente (o por debajo) con Bloopa pero que muera Bloopa si Mario salta encima de él. Añádelo a la escena.

Ahora añade el siguiente comportamiento a Bloopa: este enemigo siempre está saltando en una determinada posición. Juega con la velocidad en la dirección Y y el factor de gravedad de Bloopa para conseguir este comportamiento.

Fin de juego al morir

Modifica el comportamiento de Mario para que, en lugar de reaparecer en un punto, se muestre una ventana con el fin de la partida como la que se muestra en la imagen. Al pulsar en "Play Again" volverá a cargarse la escena de inicio. Puedes tomar como base cualquier escena de fin de juego que viene con los ejemplos de Quintus.

Fin de juego al ganar

Crea un nuevo `Sprite` llamado `Princess` para representar la meta del juego. Usa el asset de la princesa que te proporcionamos. Colócalo al final de la escena y haz que cuando Mario la toque se muestre una ventana (similar a la de muerte) pero que indique que has ganado. Para ello haz que el sprite de la princesa se comporte como un sensor (también llamado *trigger* en otros motores de juegos) y así pueda detectar la colisión con Mario.

Menú de inicio

Crea la escena para el menú de inicio usando el asset llamado `mainTitle.png`. Haz que al pulsar la tecla Intro (o al hacer click sobre esta escena) comencemos el juego. Haz que tras ganar o perder volvamos al menú de inicio.

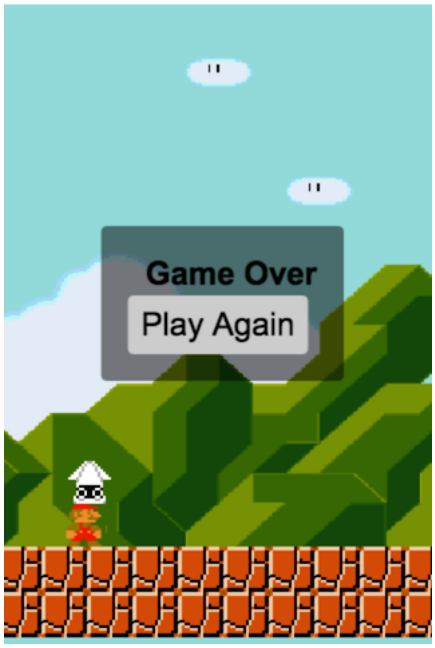

Figure 3: La ventana que aparece una vez que muere Mario

Animaciones

Usa el sistema de animaciones de Quintus (que está en el módulo `Anim`) para hacer que tanto Mario como los enemigos utilicen las animaciones adecuadas. Para Mario ten en cuenta que el componente `platformerControls` da información de la dirección en la que está mirando y que lanza eventos sobre cuando salta y cuando llega al suelo. Para los enemigos (y también para Mario) ten en cuenta que puedes hacer que las animaciones emitan un evento al terminar, de modo que las animaciones de morir pueden emitir un evento que destruya el objeto solo cuando la animación termine.

Monedas

Haz que Mario pueda recoger monedas a medida que colisiona con ellas en el escenario. Al cogerlas, la moneda sale hacia arriba y luego desaparece. Para hacer esto usa una animación por keyframes o *tween*, por lo que deberás añadir el componente `tween` a la moneda. Para guardar el número de monedas usaremos el objeto `Q.state` que proporciona Quintus. Recuerda que el número de monedas se inicia a 0 cada vez que carguemos el escenario del juego. Añade unas cuantas monedas a la escena desde código.

Crea una escena que representa el HUD, es decir, la ventana superpuesta que

muestra información de las monedas que lleva recogidas Mario. Haz que este objeto quede a la escucha de los eventos que `Q.state` genera al cambiar el número de monedas.

Componente para los enemigos

Como has podido ver, los dos enemigos tienen código repetido: cómo se comportan cuando colisionan con Mario y qué animaciones han de ejecutar en cada momento. Por este motivo, vamos a sacar este comportamiento a un componente que podamos reutilizar entre enemigos.

Crea el componente `defaultEnemy` (clase que extiende de `Component`) para crear el comportamiento por defecto de todos los enemigos al chocar con Mario. Elimina el código repetido de Goomba y Bloopa y añádeles, en su lugar, este componente. Recuerda que un componente interactúa con el gameobject al que se encuentra unido a través del atributo `entity` pero que las funciones que añadas a la sección `extend` se ejecutarán directamente sobre el gameobject.

Sonidos

Esta última sección la consideraremos como **opcional** ya que el audio en HTML5 da muchos problemas. Asegúrate de cargar correctamente todos los assets de audio antes de ejecutar la escena. Añade el módulo `Audio` para poder incluir sonidos en nuestro juego (por favor, en el laboratorio usad auriculares). Haz que el menú de inicio comience con la música principal (`main_title`). Pararemos este sonido al terminar el nivel para poner el sonido adecuado (el sonido que indica que hemos ganado o el sonido que indica que hemos perdido). Añade también el sonido de las monedas al cogerlas.

Mejoras y ampliaciones

De manera opcional se pueden añadir todas las mejoras y ampliaciones que queramos, tanto de las incluidas en el juego original como mecánicas propias. Algunas posibles ampliaciones serían las siguientes:

- **Vidas**: El juego no termina inmediatamente sino Mario tiene varias vidas. Cada vez que muere, Mario vuelve a aparecer al principio del nivel.
- **Nuevos enemigos**: Crea nuevos tipos de enemigos inspirados (o no) en los originales del Super Mario Bros.
- **Nuevos elementos**: Añade nuevos elementos y mecánicas al juego como las setas para crecer, las flores que hacen que Mario pueda disparar, plataformas que podemos romper y que esconden nuevos elementos...

- **Nuevos niveles**: Crea nuevos niveles usando el editor de niveles Tiled. Toma como base el que se proporciona junto con el juego y modifícalo añadiendo nuevos elementos.
- ...

Entrega

La entrega consistirá en un archivo ZIP que contenga una carpeta con todos los archivos necesarios para ejecutar el juego, **así como un breve documento con las mecánicas implementadas** (al menos hay que incluir las descritas en el apartado de desarrollo de la práctica). El nombre del archivo ha de ser: *Apellido1Apellido2Nombre.zip*