Práctica 2: Tapper

Guillermo Jiménez Díaz, Pedro A. González Calero, Juan A. Recio García

Entrega: 3 de abril de 2018, 23.55h

1 Introducción

En esta práctica vamos a desarrollar una simplificación de un videojuego clásico a partir del Alien Invasion (mod del *Space Invaders*) explicado en las clases teóricas y descrito en el libro *Professional HTML5 Mobile Game Development*, de Pascal Rettig. El código fuente completo del que partir se encuentra en el Campus Virtual.

2 Tapper

Tapper es un videojuego de género arcade publicado por Bally Midway en 1983. El objetivo del juego es servir cervezas a todos los clientes antes de que les hagamos perder la paciencia. Además, es necesario recoger las jarras vacías que nos van lanzando a medida que sacian su sed. Puedes jugar una demo de este juego aquí.

3 Desarrollo de la práctica

Para desarrollar este juego se proporciona, a través del Campus Virtual, un archivo zip que contiene la carpeta de directorios que tenéis que utilizar, con los siguientes contenidos:

- `src/engine.js`: Versión del motor de juego del que partiréis para realizar vuestra práctica.

Figure 1: Tapper

- `src/game.js`: Código del Alien Invaders. Utiliza como ayuda este código para crear el tapper.
- `img/spritesTapper.png`: Una *spritesheet* con los sprites necesario para implementar el juego. Tendréis que sustituirlo por el que se usa en Alien Invaders.
- `img/sprite.json`: Contiene información para cargar los sprites de la spritesheet anterior. Podéis usar el contenido de este archivo para cargar los sprites tal y como se hace en el videojuego de partida (Alien Invaders).

Comienza leyendo el código de la carpeta `src`. Te ayudará a entender de qué dispones en el motor y de cómo usarlo para crear el juego.

No es necesario implementar todas las mecánicas del juego original sino que basta con que se implementen las necesarias para tener un juego cerrado. **Se recomienda realizar las mecánicas principales en el siguiente orden.**

3.1 Escenario de fondo

Sustituye la carga del archivo de sprites del Alien Invaders por el del Tapper (esto se hace en la clase `SpriteSheet` del motor). A continuación configura correctamente la variable `sprites` que aparece al principio de `game.js` para que cargue los sprites del Tapper. Haz que se comience ejecutando el método `playGame` (en lugar de comenzar por el `startGame`) modificando la llamada al método `Game.initialize` que aparece al final del archivo `game.js`. Finalmente crea en el método `playGame` una primera capa (de tipo `GameBoard`) que tan solo contenga un sprite con la imagen de fondo. Para ello, crea un objeto que hereda de `Sprite` y que lo único que haga sea dibujar la imagen del fondo.

3.2 Movimiento del jugador

Crea la clase que `Player`, que hereda de `Sprite`. Implementa su método `step`, que hace que el camarero se mueva en cuatro posiciones fijas de manera *discreta*, es decir, moviéndose exactamente una posición en la dirección indicada por el jugador. Añade las teclas Arriba (38), Abajo (40) y Espacio (32) a las teclas que el motor controla. El movimiento ha de ser cíclico, de modo que al llegar a la parte superior vuelve a aparecer por la inferior y viceversa. Puedes usar las siguientes posiciones para definir dónde se mueve el jugador:

```
{x:325, y:90},
```

```
{x:357, y:185},
{x:389, y:281},
{x:421, y:377}
```

Por último, crea en el `playGame` una segunda capa en la que se va a desarrollar el juego (distinta de la del fondo del juego) y añade al jugador a dicha capa.

3.3 Lanzando cervezas

Crea la clase `Beer`, que hereda de `Sprite` y que hace que una cerveza llena se mueva de derecha a izquierda. Crea esta clase de modo que puedas iniciarla en distintas posiciones y que se mueva a distinta velocidad. Añade una cerveza a la capa en la que está el jugador para comprobar que se mueve adecuadamente.

Haz que el jugador cree cervezas al pulsar la tecla Espacio. Para ello crea clones de cervezas[1] a medida que pulsas esta tecla. Te recomendamos que al crear la cerveza la desplaces hacia la izquierda una distancia equivalente al ancho del sprite (si no, es probable que interfiera con mecánicas que haremos más adelante.)

3.4 Clientes y colisiones

Crea la clase `Client` que hereda de Sprite y que hace que un cliente se mueva de izquierda a derecha. Al igual que la cerveza, crea esta clase de modo que puedas iniciarla en distintas posiciones, con distintos sprites (opcional) y que se mueva a distinta velocidad. Añade un cliente a la capa en la que está el jugador para comprobar que se mueve adecuadamente.

Haz que el cliente sea eliminado cuando colisione con una cerveza. Elimina también la cerveza al producirse esta colisión.

3.5 Jarras vacías

Implementa las jarras vacías. Para ello puedes decidir entre una de las siguientes opciones:

[1] Para crear un clon de un objeto en JavaScript podemos usar el método `Object.Create` de JavaScript.

- Crea la clase `Glass` e impleméntala de manera similar a la clase `Beer` pero haciendo que se mueva de izquierda a derecha.
- Modifica la clase `Beer` para que pueda estar en dos estados (llena o vacía). Haz que en cada estado tenga un aspecto y comportamiento diferentes.

Haz que el cliente genere una jarra vacía cuando la jarra de cerveza colisione con él. Haz también que la jarra vacía desaparezca cuando colisiones con el `Player` (date cuenta que solo han de desaparecer las jarras vacías, no las cervezas).

3.6 Llegando al final de la barra

Crea la clase `DeadZone`, que representa un rectángulo o *trigger* que hace que clientes, jarras vacías y llenas sean destruidas al colisionar con él. Coloca una `DeadZone` a cada extremo de la barra y comprueba su funcionamiento. Para poder depurar te recomendamos que le implementes un método `draw` que dibuje un rectángulo usando las instrucciones primitivas del Canvas de HTML5[2]. Al colocar las `DeadZone` tendrás que tener cuidado y comprobar que cuando el `Player` crea una cerveza, ésta no se destruye inmediatamente.

3.7 Generadores de clientes

Para hacer un nivel completo y tener la forma de crear múltiples niveles con distintos clientes vamos a crear una clase `Spawner` que contiene la lógica para crear clientes en una determinada barra del bar. Haz que este objeto se pueda configurar para que se generen un número concreto de clientes de un determinado tipo, a una frecuencia de creación fija y con un determinado retardo con respecto a la creación del primer cliente (para que no salgan clientes en todas las barras a la vez).

Para ello os recomendamos que utilicéis un patrón `Prototype`. El `Spawner` es inicializado con un objeto prototípico (un `Cliente`) que iremos clonando y añadiendo a la capa del juego con una determinada frecuencia. **No debes usar el método `setInterval` para decidir cuándo clonar el objeto prototipo**[3]. Si lo haces, verás que el juego se comporta de manera extraña si mandas la ventana del navegador a segundo plano.

[2] Aquí tienes un poco de ayuda para dibujar un rectángulo en el canvas.

[3] Este motor no usa `setInterval` para implementar el bucle de juego sino un polyfill llamado `requestAnimationFrame`. Si usas el `setTimeout` entonces se producirán comportamientos extraños en la ejecución del bucle.

3.8 GameManager y condiciones de finalización

Crea un `GameManager`, es decir, un objeto *Singleton*[4] que se encargará de comprobar el estado en el que se encuentra el juego y de decidir si hemos ganado o perdido.

El jugador gana si se cumplen estas dos condiciones:

- No quedan clientes a los que servir. El número de clientes es fijo para un nivel y se conoce a priori.
- No quedan jarras vacías que recoger.

El jugador pierde si se cumple alguna de estas condiciones:

- Algún cliente llega al extremo derecho de la barra.
- Alguna jarra vacía llega al extremo derecho de la barra.
- Alguna cerveza llena llega al extremo izquierdo de la barra.

El `GameManager` es el responsable de centralizar toda la información del juego y, por tanto, de decidir cuándo se termina (porque se ha ganado o se ha perdido):

- Los `Spawners` han de avisarle al principio del juego de cuántos clientes van a generar.
- Cada vez que generemos jarras vacías deberemos avisar al `GameManager` para que lleve la cuenta.
- Cada vez que sirvamos a un cliente deberemos avisar al `GameManager` para que sepa cuántos clientes lleva servidos.
- Cada vez que una jarra caiga o un cliente llegue al extremo de la barra deberemos avisar al `GameManager`.

De momento solo haz que el `GameManager` muestre por consola si se ha ganado o se ha perdido. En la última sección lo implementaremos adecuadamente.

3.9 Menús y condiciones de finalización

Crea la pantalla de títulos inicial (usando la clase `TitleScreen` que proporciona el motor), con el título del juego y que nos indique qué tecla hay que pulsar para empezar en el método `startGame`. Vuelve a cambiar el método `Game.initialize` para que ahora el juego comience ejecutando el `startGame`.

[4] Fíjate en cómo se ha creado la variable `Game` del `engine.js`.

Crea otra pantalla de títulos para mostrar el final del juego cuando hemos perdido la partida y otra para cuando la hayamos ganado.

Haz que desde las pantallas de fin de partida podamos volver a comenzar una nueva partida. Como recomendación para implementar esto modifica el motor (`engine.js`) de modo que podamos tener inicialmente todas las capas (`GameBoard`) creadas y que las podamos activar y desactivar cuando queramos. Una capa desactivada no ejecutará ni su método `step` ni su método `draw`.

Haz que el `GameManager` se encargue de activar una u otra pantalla de finalización de acuerdo a las condiciones que explicamos en el apartado anterior.

4 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. En la carpeta de material también te hemos dejado un archivo con otros sprites que puedes usar para modificar el juego[5]. Algunas posibles ampliaciones serían las siguientes:

- **Vidas**: El juego no termina inmediatamente cuando un cliente llega al final de la barra o una jarra (llena o vacía) se cae sino que el jugador tiene de varias vidas. Cada vez que muere, el nivel se reinicia.
- **Puntos**: El jugador gana 50 puntos cada vez que sirve a un cliente y gana 100 puntos extra cuando recoge una jarra vacía. El menú principal muestra la máxima puntuación conseguida en una sesión de juego.
- **Propinas**: Los clientes pueden dejar propina (aleatoriamente) encima de la barra al ser servidos. Esto implica que el camarero ha de ir a recoger la propina para ganar los 1500 puntos que proporciona. En este caso, el jugador ha de poder moverse a izquierda y derecha de una barra pero no puede servir cervezas en ese momento. Si pulsamos arriba o abajo volvemos al comportamiento normal de movimiento implementado en la parte obligatoria de la práctica.
- **Comportamiento de los cliente**: El comportamiento de los clientes que hemos creado es una simplificación del juego original. En realidad, el comportamiento de los clientes es el siguiente: cuando reciben una cerveza, el cliente se mueve hacia atrás en la barra durante unos segundos (está bebiendo la cerveza). Si en ese movimiento sale por la puerta de la izquierda entonces se considera que el cliente ha sido servido. Si no llega a salir por la puerta durante ese tiempo que

[5]Los sprites pertenecen al juego MiniTapper.

se mueve hacia atrás entonces se detiene y vuelve a moverse hacia la derecha de la barra. Ten en cuenta que cuando el cliente está bebiendo no ha de coger las cervezas llenas que le lleguen.
- **Niveles**: Haz distintos niveles con distintos patrones de comportamiento de los generadores de clientes. Permite definir estos patrones de generación desde un archivo de datos.
- ...

5 Entrega

5.1 Entrega

La entrega se realizará a través de Github. En el Campus Virtual os dejaremos material de referencia para trabajar con Github.

La entrega consistirá en indicar en la entrega asociada a la práctica del Campus Virtual:

- Un enlace al repositorio de Github en el que se encuentra el proyecto completo.
- El hash del commit que indica la versión del proyecto que se quiere que sea evaluada.
- Un enlace a la web en la que está alojado el videojuego.