Práctica 2: Frogger

Guillermo Jiménez Díaz, Pedro A. González Calero

Entrega: 29 de marzo de 2016, 23.55h

1 Introducción

En esta práctica vamos a desarrollar un videojuego completo 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 puede descargar desde esta página o desde su repositorio en Github.

2 Frogger

Frogger es un videojuego clásico, de género arcade, publicado por Sega en 1981. El objetivo del juego es guiar una rana hasta su hogar. Para hacerlo, la rana debe evitar coches mientras cruza una carretera congestionada y luego cruzar un río lleno de peligros. El jugador hábil obtendrá puntos adicionales a lo largo del camino. Puedes jugar una demo de este juego aquí. Las principales mecánicas están descritas en esta página.

Aunque el juego tiene ya sus años hace poco se creó una versión modificada del mismo, llamada Crossy Road, y que ha causado furor (se estima que generó 10 millones de dólares en tres meses desde su release)

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.

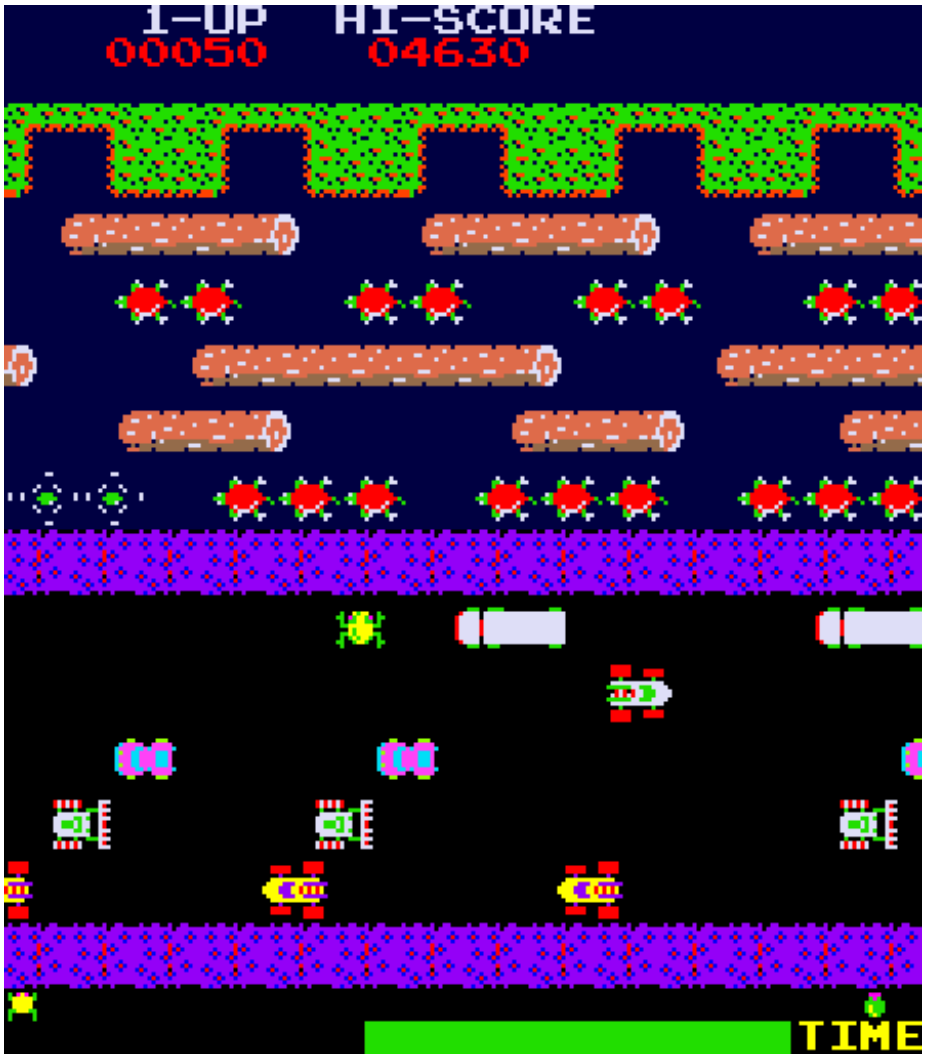

Figure 1: Frogger

- `src/game.js`: Código del Alien Invaders. Utiliza como ayuda este código para crear el frogger.
- `img/spritesFrogger.png`: Una *spritesheet* con los sprites necesario para implementar el juego.
- `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 Frogger (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 Frogger. 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 (`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 de la rana

Crea la clase que `Frog`, que hereda de `Sprite`. Implementa su método `step`, que hace que la rana se mueva en las cuatro direcciones de manera *discreta*, es decir, moviéndose exactamente una casilla en la dirección indicada por el jugador (cada casilla es de 48x48, como el sprite de la rana). Añade las teclas Arriba (38) y Abajo (40) a las teclas que el motor controla. Inicialmente no es necesario que la rana se mueva de manera fluida sino que basta con que se mueva a "saltos" (de casilla en casilla). Para hacer que se mueva de una manera más fluida puedes añadir un retardo en el procesado de la tecla pulsada en el `step`. **No uses setTimeout**[1]: deberás ir calculando el tiempo que ha pasado usando el `dt` que se pasa como parámetro del método `step`. Controla también que no se salga de los límites del escenario (el tamaño del juego está disponible en la clase `Game`).

[1] 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.

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 a ésta la rana.

3.3 Coches

Crea la clase `Car`, que hereda de `Sprite` y que hace que un coche se mueva en una dirección, en una fila, de un extremo al otro de la pantalla. Crea esta clase de modo que puedas crear coches con distinto aspecto, que se mueven en sentidos distintos, en distintas filas y a distinta velocidad. Haz que los coches se eliminen de la capa al salir por los bordes de pantalla (recuerda que los límites del juego están en la clase `Game`). Añade algunos coches a la capa en la que está la rana para comprobar que se mueven adecuadamente.

3.4 Colisiones

Haz que la rana sea eliminada de la capa al ser atropellada por un coche comprobando las colisiones entre los coches y la rana. No es necesario crear ningún tipo de animación en este momento.

3.5 Troncos

Crea la clase `Log`, que hereda de Sprite, para representar los troncos que hay en la parte superior de la pantalla. Su comportamiento básico será similar al implementado para los coches. Añade varios troncos a la capa del juego para poder probar el funcionamiento de los troncos.

Posteriormente, haz que la rana se desplace a la misma velocidad y en la misma dirección que el tronco si la rana está colisionando con el tronco. Para implementar esto puedes hacer lo siguiente:

1. Añade una velocidad horizontal `vx` a la rana. Inicialmente vale 0 pero tenla siempre en cuenta a la hora de actualizar la posición de la rana en el método `step`.
2. Añade un método `onLog (vLog)` a la rana que será llamado cuando el tronco colisiona con la rana. Este método cambiará la velocidad horizontal de la rana y la pondrá al valor del tronco. Así, la rana se desplazará horizontalmente al estar sobre el tronco.
3. El sistema de colisiones es muy rudimentario por lo que no tenemos forma de saber cuándo la rana ha dejado de colisionar con el tronco (y, por tanto, poner su `vx` a 0). En este caso, la rana se seguirá desplazando aunque se haya bajado del tronco. La forma más sencilla de arreglar este problema es inicializar a 0 `vx` siempre al final del `step`.

Si la rana aparece pintada por debajo del tronco cambia el orden en el que los añades a la capa. Tendrás que tener en cuenta esto un poco más adelante.

3.6 Agua

Crea la clase `Water`, que hereda de `Sprite`, que hace que la rana muera al colisionar con esta franja de agua (similar a lo implementado para los coches). Puedes reimplementar su método `draw` para que no dibuje nada (es decir, que sea un sprite invisible)

Haz que la rana no muera si está sobre un tronco. Para ello puedes hacer algo similar a lo realizado con la velocidad horizontal de la rana cuando está sobre el tronco. ¡Cuidado! El orden en el que los objetos se añaden a la escena (agua, troncos y rana) afectan a la correcta ejecución de este apartado.

3.7 Animación de muerte

Crea la clase `Death`, que hereda de `Sprite` y que representa un sprite animado que aparece en el lugar en el que la rana ha muerto (similar a las explosiones del juego original).

3.8 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`.

Crea otra pantalla de títulos para mostrar el final del juego cuando la rana muere (hemos perdido la partida).

Finalmente, crea una clase `Home` (similar a `Water`) que representa la meta a la que tiene que llegar la rana (situada en la parte superior de la pantalla. Este objeto mostrará una pantalla de títulos indicando que hemos ganado la partida cuando la rana colisione con ella.

Haz que desde las pantallas de fin de partida podamos volver a comenzar una nueva partida. Como recomendación para implementar esto modificad 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`.

3.9 Generadores de coches y troncos

Para hacer un nivel completo y tener la forma de crear múltiples niveles con distintos coches y troncos vamos a crear una clase `Spawner` que contiene la lógica para crear coches o troncos en una determinada fila del juego. Haz que este objeto se pueda configurar para que genere coches o troncos con unas determinadas características y a una frecuencia de creación fija.

Para ello os recomendamos que utilicéis un patrón `Prototype`. El `Spawner` es inicializado con un objeto prototípico (un `Log` o un `Car`) que iremos clonando [2] y añadiendo a la capa del juego con una determinada frecuencia. Recuerda que no debes usar el método `setInterval` para decidir cuándo clonar el objeto prototipo.

Utilizando los `Spawners` verás que la rana se dibuja por debajo de los troncos cuando están encima de ellos. Esto se debe a que los sprites se dibujan en el orden en el que se añaden en la capa (`GameBoard`) y los objetos creados por el `Spawner` se añaden después de la rana (que se creó al principio). Para resolver este problema puedes implementar alguna de las siguientes alternativas:

- Modifica el motor para que se pueden añadir objetos "al principio" de la lista de objetos de una capa. De esta forma, los que están al principio se dibujarán antes.
- Modifica el motor para que los Sprites tengan un `zIndex`. Los sprites se ordenarán en la capa de acuerdo a este `zIndex`, de modo que se pintarán antes los que tengan un valor de `zIndex` menor.

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. Algunas posibles ampliaciones serían las siguientes:

- **Vidas**: El juego no termina inmediatamente sino que la rana tiene de varias vidas para poder llegar a la parte superior de la pantalla. Cada vez que muere, la rana vuelve a aparecer en la parte inferior de la pantalla.
- **Tiempo**: La rana tiene un tiempo máximo para llegar a la parte superior. Si se acaba el tiempo entonces pierde una vida.
- **Puntos**: La rana gana 10 puntos cada vez que avanza hacia la parte superior. Si consigue alcanzar la parte superior, gana 100 puntos extra. Una vez que llega aquí, la rana vuelve automáticamente a la parte inferior para poder sumar más puntos. El menú principal muestra la máxima puntuación conseguida en una sesión de juego.

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

- **Insectos**: Haz que aparezcan en el escenario algunos insectos de manera aleatoria. Si la rana los captura entonces ganará puntos extra.
- **Serpientes**: La serpiente puede aparecer en ocasiones en la parte intermedia entre la carretera y la zona de agua. Si la rana colisiona con ella, morirá.
- **Generadores por patrones variables**: Haz que los generadores de coches y troncos permitan definir patrones de creación más variables. Haz que, a medida que la rana llega a la parte superior, los patrones cambien, haciendo cada vez más difícil el juego. Permite definir estos patrones de generación desde un archivo de datos.
- ...

5 Entrega

La entrega consistirá en un archivo ZIP con nombre *Apellido1Apellido2Nombre.zip* que contenga una carpeta con:

- Todos los archivos necesarios para ejecutar el juego
- Un breve documento con las mecánicas implementadas (al menos hay que incluir las descritas en el apartado de desarrollo de la práctica).