Juegos - Ejercicios

Índice

1 Juego de coches sencillo....2
2 Panj....2
3 Clon del Tapper....2
4 Clon del Frogger....3

1. Juego de coches sencillo

Vamos a ver un ejemplo de juego básico implementado en MIDP 2.0. Este juego se encuentra ya implementado en el directorio `Juego` de las plantillas de la sesión.

En el juego podemos encontrar los siguientes elementos:

- Tenemos un *sprite* de un coche que se mueve por la pantalla. Se utiliza como *sprite* la imagen `coche.png`.
- En el ciclo del juego se lee la entrada del usuario, y según ésta movemos el *sprite* por la pantalla (en las direcciones izquierda, derecha, arriba y abajo), y actualizamos los gráficos.
- Tenemos un fondo construido mediante un objeto `TiledLayer`. En la imagen `fondo.png` tenemos una serie de elementos con los que construir el fondo de la carretera. Utilizamos el fichero `datos` donde tenemos codificado un posible escenario. De este fichero leemos el índice del elemento que se debe mostrar en cada celda del fondo. Se encuentra codificado de la siguiente forma:

```
<ancho:int> <alto:int>
      <celda_1_1:byte> <celda_1_2:byte> ... <celda_1_ancho>
      <celda_2_1:byte> <celda_2_2:byte> ... <celda_2_ancho>
      ...
      <celda_alto_1:byte> <celda_alto_2:byte> ...
<celda_alto_ancho>
```

  Leemos estos datos utilizando un `DataInputStream`. Dibujamos este fondo como capa en la pantalla, y en cada iteración lo vamos desplazando hacia abajo para causar el efecto de que el coche avanza por la carretera.
- Detección de colisiones con nuestro coche y el fondo. De esta forma podremos chocar con los bordes de la carretera, teniendo que evitar salirnos de estos márgenes. Cuando chocamos simplemente se muestra una alerta que nos avisa de que hemos chocado y se vuelva a iniciar el juego.

2. Panj

Ejemplo de juego completo. En las plantillas de la sesión se incluye un juego completo como ejemplo en el directorio `Panj`. Consultar el código fuente de este juego y probarlo.

3. Clon del Tapper

Vamos a implementar una versión del clásico *Tapper*. En el directorio `UATapper` de las plantillas de la sesión se encuentra una base sobre la que se puede implementar el juego. En esta plantilla tenemos:

- Recursos necesarios: imágenes y fichero con los datos de las fases.

- Clase con los datos del juego (`CommonData`). En ella tenemos toda la información acerca del tamaño de los *sprites*, las secuencias de *frames* de las animaciones, sus coordenadas iniciales, etc.
- Clase del MIDlet principal (`MIDletTapper`).
- Pantalla *splash* (`SplashScreen`).
- Clase para la gestión de los recursos (`Resources`).
- Clase que implementa un ciclo de juego genérico (`GameEngine`).
- Pantalla de titulo (`TitleScene`) e interfaz para las escenas del juego (`Scene`).
- Pantalla de juego (`GameScene`) vacía. Deberemos implementar la lógica y la presentación del juego en esta clase.

Debemos:

a) Añadir un fondo a la escena. El fondo ya está cargado, según la fase del juego, en un `TiledLayer`. Solo hay que invocar su función de renderizado.

b) Añadir el *sprite* del barman. Hacer que el *sprite* se mueva por la pantalla como respuesta a las pulsaciones del cursor. El desplazamiento en horizontal se realizará solamente a lo largo de las barras que haya en pantalla, mientras que el vertical se producirá solamente cuando el barman esté en el extremo derecho de las barras.

c) Añadir los clientes a la escena. Los clientes aparecen aleatoriamente por la izquierda de alguna barra y se mueven hacia la derecha. Utilizar la información de `StageData` para controlar la frecuencia de aparición de clientes y su velocidad.

d) Añadir los sprites de las cervezas. Estas pueden estar llenas o vacias. Las llenas se mueven desde el barman hacia la izquierda, hasta que llegan a un cliente (o al final de la barra y se rompen). Las vacías se mueven desde el cliente hacia la derecha, hasta que llegan al barman, o bien a al extremo derecho de la barra y se rompen.

e) Implementar el resto de funcionalidades del juego:

- Comprobar colisiones entre cervezas vacias y barman, y colisiones entre cervezas llenas y clientes.
- Comprobar si las cervezas llegan al extremo de alguna barra sin ser cogidas por nadie. Cuando esto suceda perderemos una vida y volverá a comenzar el nivel. En el caso de que no queden vidas, terminará el juego.
- Comprobar si las cervezas puestas han alcanzado el número de cervezas de la fase correspondiente, para cambiar a la siguiente fase. Si fuese la última, podríamos hacer que vuelva al comienzo del juego (primera fase).

4. Clon del Frogger

Vamos a implementar un clon del clásico *Frogger*. En el directorio `Cochedrilo` de las plantillas de la sesión se encuentra una base sobre la que se puede implementar el juego. En esta plantilla tenemos:

- Recursos necesarios: imágenes y fichero con los datos de las fases.
- Clase con los datos del juego (`CommonData`). En ella tenemos toda la información acerca del tamaño de los *sprites*, las secuencias de *frames* de las animaciones, sus coordenadas iniciales, etc.
- Clase del MIDlet principal (`MIDletCDrilo`).
- Pantalla *splash* (`SplashScreen`).
- Clase para la gestión de los recursos (`Resources`).
- Clase que implementa un ciclo de juego genérico (`GameEngine`).
- Pantalla de titulo (`TitleScene`) e interfaz para las escenas del juego (`Scene`).
- Pantalla de juego (`GameScene`) vacía. Deberemos implementar la lógica y la presentación del juego en esta clase.

Debemos:

a) Crear las estructuras de datos necesarias para cargar la información sobre las fases. Los datos de cada fase son los siguientes:

- Título de la fase
- Carriles de la carretera. Para cada carril tendremos los siguientes datos:
  - Velocidad de los coches que circulan por él.
  - Separación que hay entre los coches del carril.
  - Tipo de coche que circula por el carril.

Estos datos se encuentras codificados en el fichero de datos de fases (`stages.dat`) de la siguiente forma:

```
<int> Numero de fases
Para cada fase
  <UTF> Titulo
  <byte> Número de carriles
  Para cada carril
    <byte> Velocidad
    <short> Separación
    <byte> Tipo de coche
```

Deberemos añadir a las clases que encapsulen estos datos métodos para deserializarlos de este fichero.

Introducir código en `Resources` para cargar esta información y añadir los datos de niveles como recurso global de nuestro juego.

b) Añadir el *sprite* de nuestro personaje. Hacer que el *sprite* se mueva por la pantalla como respuesta a las pulsaciones del cursor.

c) Añadir un fondo a la escena. El fondo se construirá utilizando un objeto `TiledLayer`. Utilizaremos la información sobre el número de carriles de la fase actual para generar este fondo.

d) Añadir los coches a la escena. Los coches deben aparecer a la izquierda y avanzar

hacia la derecha. En cada carril los coches se generarán de distinta forma. Cada carril tiene los siguientes atributos:

- Velocidad: Los coches que circulen en dicho carril irán a la velocidad indicada. La velocidad se mide en pixeles que avanzan los coches en cada *tick*.
- Separación: El espacio en pixels que queda entre un coche y el siguiente que aparece. Podemos implementar esto creando un contador para cada carril, que controle en qué momento debe hacer aparecer un nuevo coche en dicho carril. Por ejemplo, si tenemos un carril cuya velocidad es 3 y la separación 80, inicializaremos el contador a 80 y en cada tick lo decrementaremos en 3 unidades. Cuando el contador llegue a 0 haremos aparecer un coche y le sumaremos 80 para volver a empezar a contar lo que queda para aparecer el siguiente coche.
- Tipo de coche: En cada carril circulará un tipo de coche distinto. Tenemos definidos 3 tipos, cada uno de ellos con distinto aspecto y tamaño. Según el tipo (0, 1 ó 2) crearemos el sprite del coche utilizando una imagen distinta.

e) Implementar el resto de funcionalidades del juego:

- Comprobar colisiones de los coches con nuestro personaje. Cuando esto suceda perderemos una vida y volverá a comenzar el nivel. En el caso de que no queden vidas, terminará el juego.
- Comprobar si el personaje ha cruzado la carretera para pasar de fase. Cuando hayamos llegado al otro lado de la carretera pasaremos a la siguiente fase. Si fuese la última, podríamos hacer que vuelva al comienzo del juego (primera fase).