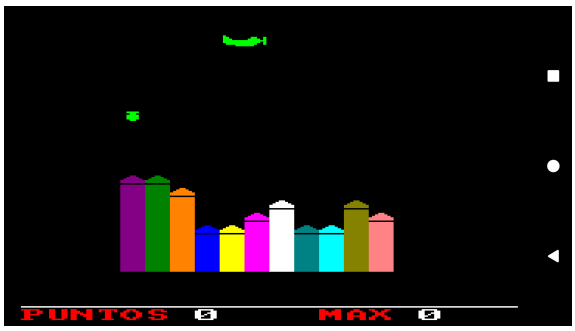
Práctica 1: Bombardero

Fecha de entrega: 18 de noviembre de 2018, 23:55

Objetivo: Iniciación a la programación en Android y Java. Arquitectura para desarrollo multiplataforma



Usted está pilotando un avión sobre una ciudad desierta y tiene que pasar sobre los edificios para aterrizar y repostar. Su avión se mueve de izquierda a derecha.

Al llegar a la derecha, el avión vuelve a salir por la izquierda, pero *más abajo*. Dispone de bombas que puede hacer caer sobre los edificios pulsando sobre la pantalla.

Cada vez que aterriza, sube la altura de los edificios y la velocidad.

Una vez disparada una bomba, ¡ya no puede disparar otra mientras no haya explosionado la primera!

El juego

La práctica consiste en la implementación de un antiguo juego desarrollado para ordenadores de 8 bits llamado "Bombardero"[1]. El jugador debe lanzar bombas desde un avión que se mueve automáticamente (de izquierda a derecha y de arriba a abajo) para destruir la ciudad (desierta) que tiene debajo y así despejar el camino para poder aterrizar. El movimiento es *discreto*, y el número de edificios y sus alturas también. Cada vez que una

[1] `https://youtu.be/mH5hi16HnvY`

bomba impacta contra un edificio, ésta destruye un número aleatorio de sus pisos aún en pie, haciéndolos desaparecer. En el juego original, durante la destrucción de un edificio el avión *se detenía*. En la implementación de la práctica el avión deberá mantenerse en movimiento.

Durante la caída de una bomba (y la destrucción, quizá parcial, de un edificio) el jugador no puede lanzar otra bomba. Si el avión, en su descenso, colisiona contra algún edificio, es destruído y la partida termina. Si el avión llega a tierra sin colisionar, el proceso se repite con edificios más altos y a mayor velocidad.

Al comenzar, el jugador debe poder elegir el nivel de dificultad, que determina la altura de los edificios, y la velocidad de simulación. Ésta determina el número de pasos (discretos) dados por el avión (y las bombas) por unidad de tiempo.

Los parámetros de funcionamiento del juego original eran:

- En cada pantalla aparecen 11 edificios (de distintos colores).
- El nivel de dificultad está entre 0 (difícil) y 5 (fácil), y se utiliza para determinar la altura de los edificios. Para una dificultad d, la altura *mínima* de cada edificio (sin contar el "tejado") es $5 - d$. A esa altura mínima se le añade un valor aleatorio entre 0 y 7. Solo si el resultado es mayor que 0 se debe incluir el tejado. Si no, en esa posición no habrá edificio.
- Al impactar sobre un edificio, una bomba destruye entre 1 y 3 bloques (el tejado cuenta como uno). Dado que en el juego original el avión se detiene durante la destrucción pero en la implementación de la práctica no debe hacerlo, se puede incrementar este número para simplificar el juego (por ejemplo, entre 2 y 4).
- El avión ocupa el ancho de dos edificios.
- El avance (discreto) del avión ocurre en el mismo instante que el descenso (discreto) de la bomba, y a la misma velocidad (una posición cada vez).
- Las bombas se dejan caer únicamente en los momentos de avance de la simulación, debajo de la parte delantera del avión tras avanzar.

La discretización de la pantalla aparece esquematizada en la figura 1. El juego original utilizaba un modo de pantalla en el que los *píxeles* no eran cuadrados, sino que tenían una relación ancho-alto de 2 a 1 (el doble de ancho que de alto).

Requisitos de la implementación

El juego se implementará en Java, y deberán generarse dos versiones diferentes, una para Android y otra para sistemas de escritorio (Windows o GNU/Linux). El código deberá estar correctamente distribuído en paquetes, ser comprensible y estar suficientemente documentado.

Para el desarrollo se hará uso de Android Studio, utilizando un único proyecto con varios *módulos*. La mayor parte del código *deberá ser común* y aparecer una única vez, compartida entre ambas versiones. Deberá existir así una *capa de abstracción* de la plataforma, que se implemente dos veces, una para Android y otra para escritorio. La implementación del juego deberá hacer uso únicamente de la capa de abstracción y de las funcionalidades del lenguaje que estén disponibles en ambas plataformas.

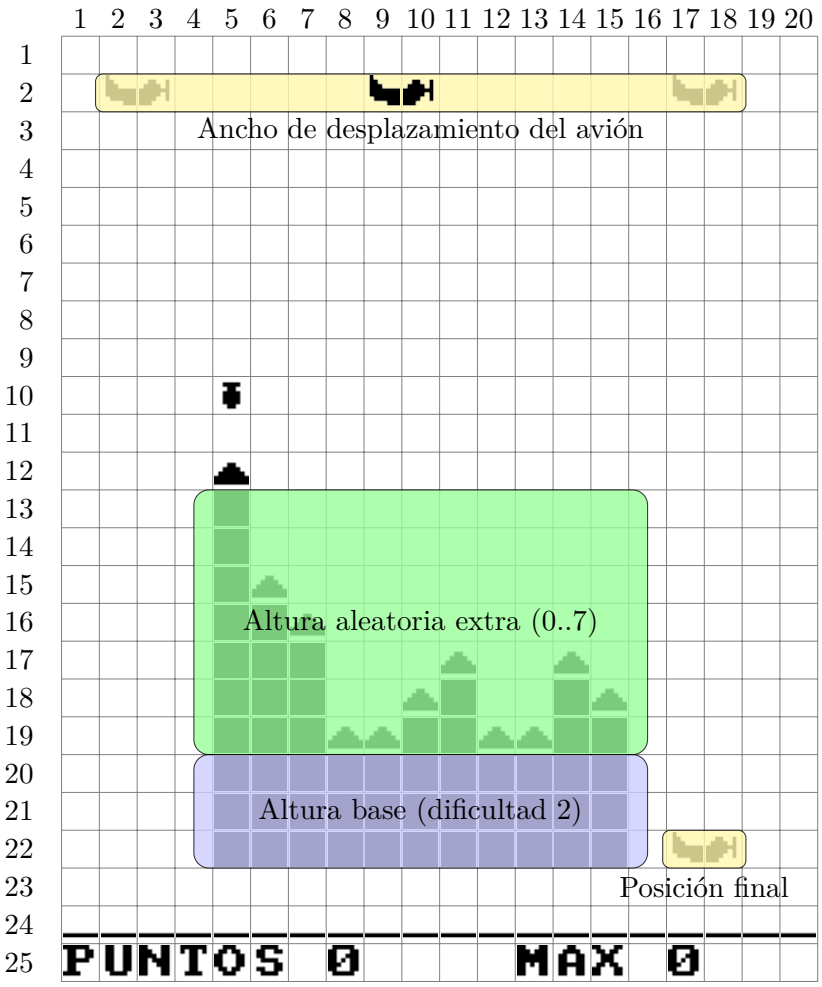


Figura 1: Distribución de los elementos en la pantalla

Dado que el punto de entrada de la aplicación es diferente en cada plataforma (en Android se necesita una `Activity` y en escritorio un `main()`) se permitirá también la existencia de dos módulos distintos (minimalistas) de "arranque del juego".

Consejos de implementación

Para abstraer la plataforma, podéis definir los siguientes interfaces:

- `Image`: envuelve una imagen de mapa de bits para ser utilizada a modo de *sprite*:
  - `int getWidth()`: devuelve el ancho de la imagen.
  - `int getHeight()`: devuelve el alto de la imagen.
- `Graphics`: proporciona las funcionalidades gráficas mínimas sobre la ventana de la aplicación:
  - `Image newImage(String name)`: carga una imagen almacenada en el contenedor de recursos de la aplicación a partir de su nombre.

    - `void clear(int color)`: borra el contenido completo de la ventana, rellenándolo con un color recibido como parámetro.
    - `void drawImage(Image image, ...)`: recibe una imagen y la muestra en la pantalla. Se pueden necesitar diferentes versiones de este método dependiendo de si se permite o no escalar la imagen, si se permite elegir qué porción de la imagen original se muestra, etcétera.
    - `int getWidth()`, `int getHeight()`: devuelven el tamaño de la ventana.

- `Input`: proporciona las funcionalidades de entrada básicas. El juego no requiere un interfaz complejo, por lo que se utiliza únicamente la pulsación sobre la pantalla (o *click* con el ratón).

    - `class TouchEvent`: clase que representa la información de un toque sobre la pantalla (o evento de ratón). Indicará el tipo (pulsación, liberación, desplazamiento), la posición y el identificador del "dedo" (o botón).
    - `List<TouchEvent> getTouchEvents()`: devuelve la lista de eventos recibidos desde la última invocación.

- `Game`: interfaz que aglutina todo lo demás. En condiciones normales, `Graphics` e `Input` serían *singleton*. Sin embargo, al ser *interfaces* y no existir en Java precompilador no es tan sencillo. El interfaz `Game` puede ser el encargado de mantener las instancias:

    - `Graphics getGraphics()`: devuelve la instancia del "motor" gráfico.
    - `Input getInput()`: devuelve la instancia del gestor de entrada.

Es posible que quieras ampliar `Game` para incorporar la idea de *estado* de la aplicación.

Al ser interfaces, observa que *no* se indica cómo se crearán las instancias. Así, por ejemplo, la clase que implemente `Graphics` para la versión de escritorio podría necesitar recibir en su constructor la ventana de la aplicación, y la versión de Android el `SurfaceView` y `AssetManager`. La puesta en marcha de la aplicación tendrá que ser diferente en cada plataforma y estar encapsulada, en la medida de lo posible, en los módulos correspondientes.

Recursos suministrados

El juego original era *en modo texto*, utilizando una tabla ASCII en el que algunos valores generaban imágenes gráficas. Como ayuda para el desarrollo, se proporcionan ficheros `.png` con *hojas de sprites* que incorporan los 256 caracteres ASCII en una matriz de 16x16 *sprites*. El primer símbolo (correspondiente al valor ASCII `0x00`) aparece en la esquina superior izquierda, y se avanza de izquierda a derecha y de arriba a abajo. Se proporciona la misma hoja en varios colores.

Los símbolos "gráficos" útiles (y sus correpondientes valores ASCII) aparecen enumerados en la tabla 1. Las letras (sin acentuar) y los dígitos tienen el valor ASCII esperado.

Como referencia (o, más bien, como curiosidad histórica) se proporciona también el código del juego original para Amstrad CPC escrito en Basic.

Símbolo	# ASCII	Uso
▲	244	Tejado de los edificios
■	143	"Pisos" de los edificios
◣	241	Lado izquierdo del avión
◢	242	Lado derecho del avión
•	252	Bomba
✕	188	Colisión 1
⊠	238	Colisión 2
▼	253	Colisión 3
_	95	Subrayado (separador barra de puntuación)

Tabla 1: Símbolos gráficos y sus valores ASCII

Partes opcionales

Siempre que los requisitos básicos de la práctica funcionen correctamente y estén bien implementados, se valorará positivamente la incorporación de características adicionales como por ejemplo:

- Inclusión de efectos de sonido.
- En la versión de escritorio:
  - Uso de pantalla completa.
  - Uso del teclado para disparar bombas y elegir el nivel de dificultad.
  - Pausa del juego (sin consumir recursos) cuando la aplicación pierda el foco.

Entrega de la práctica

La práctica debe entregarse utilizando el mecanismo de entregas del campus virtual, no más tarde de la fecha y hora indicada en la cabecera de la práctica.

Sólo un miembro del grupo deberá realizar la entrega, que consistirá en un archivo `.zip` con el proyecto completo de Android Studio eliminando los ficheros temporales. Se añadirá también un fichero `alumnos.txt` con el nombre completo de los alumnos.

Si desarrolláis alguna ampliación, deberéis incluir (dentro del `.zip`) un fichero llamado `ampliaciones.txt` describiéndolas.

El `.zip` deberá tener como nombre los nombres de los integrantes del grupo con la forma `Apellidos1_Nombre1-Apellidos2_Nombre2.zip`. Por ejemplo para el grupo formado por Miguel de Cervantes Saavedra y Santiago Ramón y Cajal, el fichero se llamará `DeCervantesSaavedra_Miguel-RamonYCajal_Santiago.zip`.

Bibliografía

- Beginning Android Games, Third Edition, Mario Zechner and J. F. DiMarzio, Apress, 2016.
- Developing games in Java, David Brackeen, Bret Barker, Lawrence Vanhelsuwe, New Riders.