Universidad de Zaragoza

VIDEOJUEGOS

Super Bomberman

Hecho por:
Jaime Ruiz-Borau Vizárraga (546751)
Patricia Lázaro Tello (554309)

Índice

1. Descripción del videojuego realizado

Como se describe en el título de la memoria, el videojuego clásico elegido para la elaboración de un clon es el **Bomberman**. Dada la elevada similitud de mecánica, enemigos y objetivos entre las distintas entregas de Bomberman, se optó por implementar la mecánica del **primer Bomberman** que salió al mercado. Sin embargo, los sprites y las imágenes empleadas para el juego son de entregas de Bomberman posteriores.

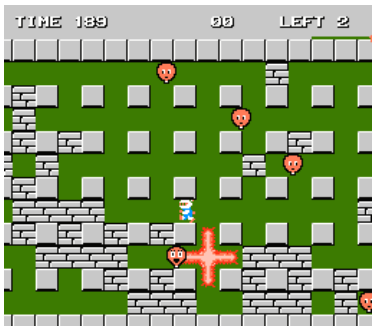

Figura 1: Bomberman clásico

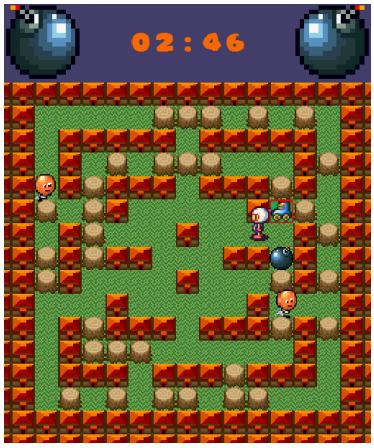

Figura 2: Clon desarrollado

La única diferencia con el original es que la puntuación no se muestra hasta el final de la partida. También se ha implementado un menú con opciones para regular el volumen de la música del juego y la posibilidad de configurar los controles del juego.

2. Detalles de la implementación en 2D

Para el juego en 2D se implementó un motor gráfico basado en un hilo principal renderizando frames a razón de 60 por segundo, mostrando los distintos sprites del juego utilizando las funciones estándar de Java para mostrar gráficos por pantalla.

El juego actualmente tiene 10 niveles generados aleatoriamente: posee unos mapas predefinidos de bloques no destruibles y dispone de forma aleatoria los enemigos y los bloques destruibles. Dichos bloques soltarán también de forma aleatoria los *power-ups* mencionados anteriormente. Se obtienen puntos por destruir enemigos, por destruir bloques y por terminar niveles, y al terminar la partida, ya sea por acabar los 10 niveles o por morir a manos de un enemigo o una bomba, se muestra la tabla de mejores puntuaciones junto a la puntuación de dicha partida.

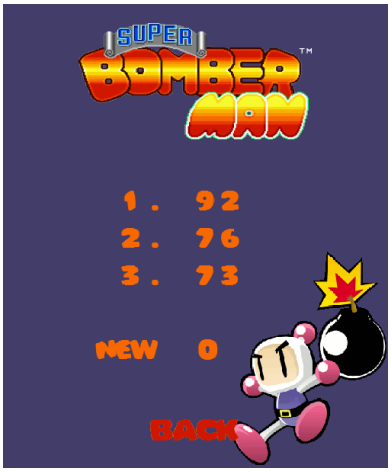

Figura 3: Pantalla de puntuación

Bomberman, el personaje que controla el jugador, puede moverse hacia arriba, abajo y hacia los lados; no puede atravesar bloques de ningún tipo y morirá si toca algún enemigo o la explosión de alguna bomba. Sin embargo, puede poner bombas para abrirse camino entre los bloques destruibles y eliminar los enemigos que salgan a su paso.

La IA de los enemigos se ha implementado siguiendo las líneas del Bomberman clásico: patrones aleatorios de desplazamiento de los enemigos. Se ha optado por no implementar una IA más compleja ya que resultaba frustrante no poder acertarle a los enemigos y resultaba muy complicado el ganar las partidas, además de que el juego original no contaba con esa IA avanzada.

Inicialmente las bombas tienen un radio de explosión de 1 casilla, sólo es posible poner una bomba a la vez en el escenario y Bomberman se mueve relativamente despacio.

Sin embargo, conforme vaya obteniendo *power ups*, el radio de las bombas cada vez es mayor, es posible poner más bombas en el escenario y Bomberman se irá moviendo más rápido, según el tipo de *power up* que recoja.

Figura 4: Power up que aumenta el radio de explosión de las bombas

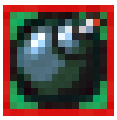

Figura 5: Power up que aumenta el número máximo de bombas en el escenario

Figura 6: Power up que aumenta la velocidad de Bomberman

El juego también contiene un menú principal desde el que se puede acceder a todos los apartados del mismo:

- Juego en 2D
- Juego en 3D
- Opciones de sonido
- Opciones de controles
- Ranking
- Créditos

Figura 7: Pantalla principal

3. Detalles de la implementación en 3D

En cuanto a la implementación del juego en 3D, se ha optado por utilizar una librería apta para el lenguaje Java; se trata de **libgdx**.

Esta librería permite cargar modelos 3D en el juego, así como utilizar y mover una cámara por el mundo 3D. Se han utilizado modelos de cubos para los bloques, enemigos, power ups y para Bomberman, variando únicamente en el color para distinguirlos. Para las bombas y sus explosiones se han utilizado esferas y cilindros, respectivamente.

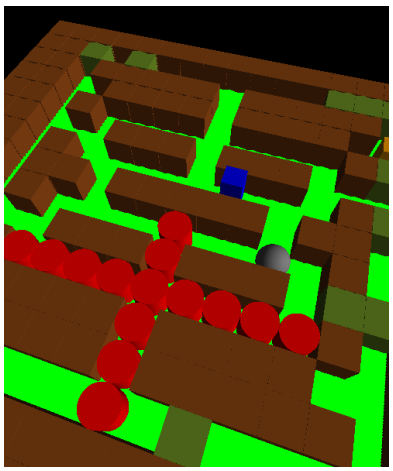

Figura 8: Pantalla del juego en 3D

Todo el sistema de juego 3D funciona exactamente igual que el 2D, únicamente cambia su aspecto visual. Sin embargo, solo se ha implementado un nivel en la versión 3D.

4. Principales dificultades encontradas

A continuación se describen las principales dificultades encontradas durante la realización del videojuego, todas ellas referidas a detalles técnicos. En la sección creativa no se encontraron dificultades, ya que el juego desarrollado es un clon y no un juego original.

- Estructura básica del juego: uno de los primeros problemas encontrados fue cómo estructurar el juego, más allá del corazón del juego. Como ambos miembros del equipo habían utilizado la herramienta GameMaker, se decidió finalmente implementar un sistema de habitaciones parecido; los objetos (las instancias que descienden de la clase Objeto) también fueron modelados de acuerdo a GameMaker, pues se consideró que era la mejor estructura que implementar.

- El paso a 3D: para el paso a 3D del juego se decidió utilizar una librería (libgdx) ya que no se tenía tiempo suficiente para implementar un motor gráfico 3D para el juego. Esto resultó en algunos problemas, ya que solo se deseaba integrar la librería en la parte 3D del juego, y no en la 2D. Algunos de estos problemas, los más notables, fueron:

  - Integración con el sistema de habitaciones realizado en el juego 2D: el juego debía funcionar igual en 3D que en 2D, y reimplementar todo para el juego 3D no era una opción. Al final, se decidió crear la habitación del juego 2D en el juego 3D, pero sin renderizarla.
  - Integración con el sistema de ventanas: también hubo problemas integrando el juego 3D en el sistema de ventanas que se tenía hasta ese momento. No se pudo integrar finalmente y se tuvo que realizar el juego 3D en otra ventana aparte. Tampoco se pudo continuar el hilo del juego una vez terminado el modo 3D, ya que la librería no funcionaba bien si se recreaba la aplicación 3D más tarde.
  - Cálculos en el juego: en un principio, se pensó en utilizar volúmenes envolventes para realizar el chequeo de colisiones de los cuerpos en el entorno y con otros cuerpos, pero se tuvo que desestimar la idea, ya que el cáclulo de colisiones de esta forma era muy pesado y no se podía ejecutar en el mismo hilo en el que se ejecutaba el rendering de la aplicación 3D (no había opción a ejecutarlo en otro hilo).

- El sistema de ayuda del juego: en versiones iniciales del juego no existía un sistema de ayuda al jugador para mover al Bomberman por la habitación. Esto hacía el juego muy complicado porque requería un control a nivel de píxel de las cajas de colisión, y hacía el juego muy frustrante y poco divertido. Para eliminar este problema se decidió incluir un sistema de ayuda que permite al jugador moverse más libremente: el programa rectifica la trayectoria del Bomberman para que se mueva en la dirección que el jugador desea con cierta libertad.

- Empaquetamiento de la aplicación: al ir a empaquetar la aplicación (crear un JAR con sus contenidos) también hubo problemas y dificultades de diversas índoles que no se habían contemplado hasta ese momento en el trabajo. A continuación se explican las más importantes:

- Cambios de ruta en el JAR: al hacer el JAR el juego no funcionaba, ya que tomaba rutas relativas fuera del JAR, cuando los archivos estaban dentro. Hubo varios problemas al respecto, ya que se utilizaban tanto archivos INI como archivos de música e imágenes (stripes de sprites) y su tratamiento era diferente.

  El principal problema era que en un JAR no hay ficheros, como en el sistema de ficheros, que es desde donde se operaba hasta ese momento, sino recursos. Para los archivos INI y las imágenes no supuso ningún problema el utilizar recursos en vez de ficheros, pero para las músicas utilizamos la librería TinySound, que requerían expresamente de ficheros. Para solucionar este contratiempo se crearon ficheros temporales en el mismo lugar donde se encontraba el JAR.

  El segundo gran problema que surgió fue que la estructura de directorios cambió al empaquetar en un JAR, y por tanto las rutas que se utilizaban ya no eran válidas. Para ello hubo que cambiar las rutas para que fueran relativas desde dentro de la carpeta "resources".
- Lentitud al cargar el JAR: en un principio se decidió empaquetar las librerías utilizadas dentro del JAR creado, pero debido a esto el juego cargaba muy lento desde el JAR (podía llegar a tardar un minuto en salir de la pantalla de carga, cuando desde el sistema de ficheros el juego tardaba menos de diez segundos en cargar).

  Después de investigar este problema, el equipo se dio cuenta de que el empaquetamiento de librerías en el JAR era el causante de la lentitud. Como las librerías estaban empaquetadas, el JAR debía extraerlas y verificarlas cada vez que se lanzaba, y la etapa de verificación era muy costosa.

  La solución a este problema fue tan sencilla como extraer las librerías dentro del JAR.

5. Módulos y paquetes del juego

5.1. Paquetes desarrollados

En este apartado se describen brevemente los paquetes que se han desarrollado a lo largo del proyecto. Por orden alfabético:

- **Graphics.d3**: Contiene una única clase, que es la encargada de toda la parte del diseño 3D del juego. Utiliza la mecánica del juego mediante otros paquetes pero mostrándolo todo en 3D.
- **Graphics.effects**: Incluye pequeñas clases con la implementación de diversos efectos visuales y algunos objetos de los menús.
- **Graphics.effects.slider**: Contiene la implementación del '*slider*' para controlar el volumen del juego en el apartado de opciones junto a una abstracción para implementar otros *sliders*. Sin embargo, en esta versión no hay otros de momento.
- **Graphics.rooms**: Incluye la clase abstracta **Room**, que representa la entidad que contiene, renderiza y gestiona los objetos del juego, así como dos clases más para cargar los recursos necesarios para cada Room antes de mostrarla y evitar así "lag"mientras se juega.
- **Graphics.rooms.controlsMenu**: Heredado de la clase Room, incluye la clase ControlsMenu que representa la room con los controles y una clase ControlRepository, donde se cargan todos los recursos necesarios para evitar posteriores tiempos de carga.
- **Graphics.rooms.credits**: Heredado de la clase Room, incluye la clase Credits que representa la room con los créditos y una clase CreditsRepository, donde se cargan todos los recursos necesarios para evitar posteriores tiempos de carga.
- **Graphics.rooms.game**: Heredado de la clase Room, incluye la clase Game que representa la room con el juego principal y una clase GameRepository, donde se cargan todos los recursos necesarios para evitar posteriores tiempos de carga.
- **Graphics.rooms.intro**: Heredado de la clase Room, incluye la clase Intro que representa la room con la intro del juego y una clase IntroRepository, donde se cargan todos los recursos necesarios para evitar posteriores tiempos de carga.
- **Graphics.rooms.mainMenu**: Heredado de la clase Room, incluye la clase MainMenu que representa la room con el menú principal y una clase MainMenuRepository, donde se cargan todos los recursos necesarios para evitar posteriores tiempos de carga.
- **Graphics.rooms.optionsMenu**: Heredado de la clase Room, incluye la clase OptionsMenu que representa la room con las opciones del juego y una clase OptionsMenuRepository, donde se cargan todos los recursos necesarios para evitar posteriores tiempos de carga.

- **Graphics.rooms.pauseMenu**: Heredado de la clase Room, incluye la clase PauseMenu que representa la room con el menú de pause y una clase PauseMenuRepository, donde se cargan todos los recursos necesarios para evitar posteriores tiempos de carga.
- **Graphics.rooms.rankMenu**: Heredado de la clase Room, incluye la clase RankMenu que representa la room con el ranking y una clase RankMenuRepository, donde se cargan todos los recursos necesarios para evitar posteriores tiempos de carga.
- **Logic**: Contiene la implementación de los elementos principales del motor del juego, tales como las variables globales, el mecanismo de entrada de inputs por teclado, la clase abstracta Objeto, la clase Sprite que contiene las subimágenes de un elemento del juego y la máquina de estados del juego, empleada para cargar las distintas rooms del mismo.
- **Logic.characters**: Implementación de clases más concretas heredadas de la clase abstracta Objecto, este paquete incluye todos aquellos elementos del juego con los que el personaje protagonista, Bomberman, puede interactuar, él mismo incluido.
- **Logic.collisions**: Contiene todo el mecanismo de colisiones en 2D gestionado mediante intersecciones a lo largo de las 4 clases que lo componen mediante cajas de colisión.
- **Logic.misc**: Contiene diversas clases que ayudan al manejo del juego pero que no tienen cabida en otro paquete. Aquí se encuentran clases como el generador de niveles aleatorio, el cargador de mapas de un fichero de texto, clases para comparar otras clases del sistema, etc.
- **Logic.misc.objectives**: Contiene la implementación abstracta y también concreta de los objetivos en los distintos niveles que se generan aleatoriamente, siendo los objetivos de los niveles dos: eliminar a todos los enemigos y llegar a las escaleras.
- **Main**: Contiene clases que se encargan de inicializar el juego.
- **Sound**: Contiene clases para gestionar la música y los efectos de sonido del juego.
- **Utils**: Similar al paquete Logic.misc, contiene clases que no tenían cabida en otro paquete, pero con un propósito más general que el de Logic.misc.

5.2. Librerías utilizadas

A continuación se listan las librerías externas que se han utilizado en la elaboración del juego y qué trabajos desarrollan dichas librerías dentro del juego.

- TinySound: es la librería responsable de reproducir la música de fondo y efectos de sonidos dentro del juego.
- Ini4j: es la librería encargada del acceso a los archivos INI. Proporciona una interfaz amigable para el acceso a archivos de este tipo y aunque este

trabajo podría haberlo hecho el equipo sin una carga de trabajo adicional considerable, se decidió apoyarse en esta librería por la simplicidad de su API.

- Libgdx: es la librería encargada de gestionar la parte 3D del juego; es decir, es el motor gráfico 3D del juego.

6. Reparto de tareas

A continuación se muestra el reparto de tareas de acuerdo a los grandes bloques del juego. No obstante, los dos miembros del equipo han participado en todas las tareas; este reparto muestra quién ha realizado más en la tarea, y no es exclusiva de una persona.

- Estructura básica del juego: Jaime Ruiz-Borau.
- Inteligencia Artificial: Patricia Lázaro.
- Control del personaje: Jaime Ruiz-Borau y Patricia Lázaro.
- Menús: Jaime Ruiz-Borau y Patricia Lázaro.
- Música: Jaime Ruiz-Borau.
- Gráficos: Jaime Ruiz-Borau y Patricia Lázaro.
- Paso a 3D: Patricia Lázaro.

Se ha optado por no incluir un cronograma del desarrollo del videojuego ya que no se ha hecho un seguimiento tan exhaustivo del desarrollo del mismo.

Referencias

[1] **Imágenes del primer Bomberman** - By Source, Fair use, [https://en.wikipedia.org/w/index.php?curid=12465479]

[2] **Enlace al repositorio de Github** - [https://github.com/EquipoJP/SuperBomberman]

[3] **Descargas del juego** - [https://github.com/EquipoJP/SuperBomberman/releases]

[4] **Página principal de Ini4j** - [http://ini4j.sourceforge.net/]

[5] **Página principal de TinySound** - [https://github.com/finnkuusisto/TinySound]

[6] **Página principal de Libgdx** - [https://libgdx.badlogicgames.com/]