PROYECTO DE PROGRAMACIÓN

AÑO ACADÉMICO: 2019-2020

ASIGNATURA: Tecnología de la Programación

PROFESOR: Juan Antonio Sánchez Laguna

ALUMNO: David Fernández Expósito

DNI: [redacted]

GRUPO: PCEO

SUBGRUPO DE PRÁCTICAS: 1.4 (PCEO)

CONVOCATORIA: Junio

ÍNDICE DE CONTENIDOS

1. DESCRIPCIÓN DE LA APLICACIÓN

1.1 OBJETIVOS

El objetivo principal de este proyecto era crear una aplicación que use el lenguaje C, utilizando el entorno de programación CodeBlocks, y haciendo uso de la biblioteca de funciones gráficas SDL a través del módulo Pantalla que fue proporcionado.

La aplicación, en este caso un videojuego, debía utilizar ficheros para guardar datos relevantes, implementar TDAs simples y contenedores, tanto con representación contigua como con representación enlazada, y tener comentado el código y usar la herramienta Doxygen para generar su documentación.

1.2 FUNCIONAMIENTO GENERAL DE LA APLICACIÓN

Como se ha comentado anteriormente, la aplicación se trata de un videojuego, que al ejecutarse muestra en primer lugar un menú principal. En él, el jugador puede elegir 3 opciones usando el ratón: “Jugar” para jugar una partida, “Ayuda” para que se muestre la explicación del juego, o “Salir” si desea salir del juego. En caso de entrar al menú de ayuda, el jugador podrá volver al menú de nuevo. Por otro lado, si el jugador ha hecho click en “Jugar”, al acabar la partida se le notificará de la puntuación obtenido y, en su caso, del récord que ha obtenido. Posteriormente, podrá pulsar la tecla ‘X’ para salir del juego.

2. MANUAL DE USUSARIO

Cuando se ejecuta el juego, aparece el menú principal del mismo. En él, el jugador podrá elegir, haciendo uso del ratón, entre: “Jugar”, “Ayuda” y “Salir”.

2.1 JUGAR PARTIDA

Si el usuario hace click sobre la opción “Jugar” comenzará inmediatamente una partida. Aparecerá el personaje (Mario) e irán apareciendo enemigos, con un número en la esquina superior izquierda de cada uno, indicando sus vidas restantes. El jugador podrá encontrar, en la esquina superior derecha de la pantalla del juego, dos mensajes, uno indicando las vidas restantes de Mario y otro indicando los puntos obtenidos hasta el momento. Las vidas iniciales son 5 y los puntos comienzan siendo 0.

2.1.1 CONTROLES

Los controles del juego son simples. Para mover a Mario, el personaje, el jugador debe hacer uso de las flechas del teclado:

Flecha “izquierda”: mueve a Mario horizontalmente a la izquierda.

Flecha “derecha”: mueve a Mario horizontalmente a la derecha.

Flecha “arriba”: hace que Mario salte.

Además, para eliminar a los enemigos que van apareciendo encima de Mario, éste puede disparar. Para ello, el jugador debe pulsar la barra espaciadora.

2.1.2 OBJETIVOS Y EXPLICACIÓN DEL JUEGO

El objetivo del jugador es eliminar al mayor número de virus posible. Este tipo de enemigo irá apareciendo cada cierto tiempo encima del personaje, y el jugador debe dispararles para eliminarlos. El ritmo de aparición de los virus aumenta conforme se van consiguiendo puntos.

Los virus disparan cada cierto tiempo, el cual se va reduciendo conforme se obtienen puntos. Si alcanzan al personaje, éste perderá una vida. También se debe tener en cuenta que, como el personaje puede saltar, si se toca a un virus también se perderá una vida,

aunque ese enemigo desaparecerá. Además, cada 10 puntos, aparece otro tipo de enemigo, una seta que persigue al personaje. El jugador debe tratar de esquivarla saltando por encima o de eliminarla saltando encima de ella, “chafándola”. La seta desaparece si toca al personaje, en cuyo caso también se pierde una vida, o si el personaje la “chafa”, en cuyo caso se gana 1 punto. Por último, cada 10 puntos, también aparece una vacuna, la cual permitirá al personaje recuperar una vida si éste la alcanza. El juego termina cuando el personaje pierde todas las vidas.

Cuando el juego termina, el jugador será informado de la puntuación obtenida en la pantalla de “game over”. Si se ha batido el récord de puntuación, también se notificará de ello y se almacenará el récord.

2.2 AYUDA

Si el jugador hace click sobre la opción “Ayuda”, aparecerá una imagen con una breve explicación del juego. El jugador podrá hacer click de nuevo en una opción “Salir” o pulsar la tecla ‘X’, ambas acciones le llevarán de nuevo al menú principal.

2.3 SALIR

Si el jugador hace click sobre la opción “Salir”, se saldrá del juego.

3. ORGANIZACIÓN DEL PROYECTO

3.1 DESCRIPCIÓN DE LOS FICHEROS QUE COMPONEN LA APLICACIÓN

- main.c: Fichero con el código principal del juego y las funciones del menú principal el de ayuda, y la función juega_partida.
- Bala.c y Bala.h: Ficheros del TDA Bala. En particular, el .c implementa las siguientes funciones: *crea_bala*, *libera_bala, mueve_bala, dibuja_bala, get_x_bala, get_y_bala* y *colision_bala.*

  Como en la sesión 7 de prácticas se pidió escribir la especificación informal del TDA Bala, se incluye a continuación:

  NOMBRE: Bala.

  DESCRIPCIÓN: Representa a una bala con unas coordenadas en el plano, velocidad en ambos ejes, una anchura y una altura, y la imagen con la que se representaría gráficamente.

  DOMINIO DE VALORES: Las coordenadas son parejas de valores de tipo real. También son números reales las velocidades de ambos ejes, el ancho y el alto. La imagen es del tipo Imagen.

  INTERFAZ PÚBLICO: consta de las siguientes funciones:

  ```
  Bala crea_bala (double x, double y, double vx, double vy);
  ```

  Esta función crea una bala con coordenadas (x,y) y velocidades vx y vy en el eje x y eje y respectivamente.

  ```
  void libera_bala (Bala b);
  ```

  Esta función libera la memoria asociada a la bala b.

  ```
  void mueve_bala (Bala b);
  ```

  Esta función mueve la bala b, tanto en el eje x como en el y, en la cantidad indicada por la velocidad indicada al crearla.

  ```
  void dibuja_bala (Bala b);
  ```

Esta función dibuja la bala b en pantalla.

```
double get_x_bala (Bala b);
```

Esta función devuelve el valor de la coordenada x de la bala b.

```
double get_y_bala (Bala b);
```

Esta función devuelve el valor de la coordenada y de la bala b.

```
int collision_bala (Bala b, double x, double y, double w, double h);
```

Esta función devuelve 1 si la bala b se solapa o "colisiona" con el rectángulo de coordeadas (x,y), anchura w y altura h. Si no colisiona, devuelve 0.

- Colisiones.c y Colisiones.h: Ficheros del módulo que trata las colisiones. En particular, el .c implementa las siguientes funciones: *dentro_rectángulo, distancia_punto_punto, solape_circunferencias* y *solape_rectángulos*.
- Ejercito.c y Ejercito.h: Ficheros del TDA Ejercito. En particular, el .c implementa las siguientes funciones: *crea_ejercito, libera_ejercito, inserta_enemigo, mueve_ejercito, dibuja_ejercito, colision_ejercito_objeto, colision_ejercito_rafaga, ejercito_dispara y colision_ejercito_personaje.*
- Enemigo.c y Enemigo.h: Ficheros del TDA Enemigo. En particular, el .c implementa las siguientes funciones: *crea_enemigo, libera_enemigo, dibuja_vidas_enemigo, dibuja_enemigo, mueve_enemigo, get_x_enemigo, get_y_enemigo, get_w_enemigo, get_h_enemigo, get_vidas_enemigo, get_tipo_enemigo, colision_enemigo, resta_vidas_enemigo* y *enemigo_dispara.*
- Pantalla.c y Pantalla.h: Ficheros de la biblioteca Pantalla proporcionada para poder realizar programas interactivos que usan gráficos.
- Personaje.c y Personaje.h: Ficheros del TDA Personaje. En particular, el .c implementa las siguientes funciones: *crea_personaje, libera_personaje, dibuja_personaje, mueve_personaje, salta_personaje, mueve_personaje_salto, suma_puntos, suma_vidas, get_x_personaje, get_y_personaje, get_w_personaje, get_h_personaje, get_vidas_personaje, get_puntos_personaje,*

personaje_dispara, personaje_activa_invulnerabilidad, personaje_comprueba_invulnerabilidad y *colision_personaje_rafaga.*

- Rafaga.c y Rafaga.h: Ficheros del TDA Rafaga. En particular, el .c implementa las siguientes funciones: *nuevo_nodo, crea_rafaga, inserta_rafaga, libera_rafaga, mueve_rafaga, dibuja_rafaga, longitud_rafaga* y *colision_rafaga.*

3.2 RELACIONES ENTRE LOS DISTINTOS MÓDULOS

Obviamente, main.c depende todos los módulos y TDA, y cada fichero ".c" depende de su respectivo fichero ".h".

Además, todos los módulos y TDA dependen de Pantalla.h, excepto el módulo Colisiones, que no depende de nadie (el .c solo depende de su propio .h). Así, las relaciones quedan de la siguiente forma (en el diagrama, A->B indica que A depende de B):

- main: depende de todos.
- Bala: depende de Pantalla y Colisiones.
- Colisiones: no depende de ningún otro módulo.
- Ejército: depende de Ráfaga, Enemigo, Pantalla y Personaje.
- Enemigo: depende de Pantalla, Colisiones, Ráfaga, Bala y Personaje.
- Personaje: depende de Pantalla, Ráfaga y Bala.
- Ráfaga: depende de Pantalla y Bala.

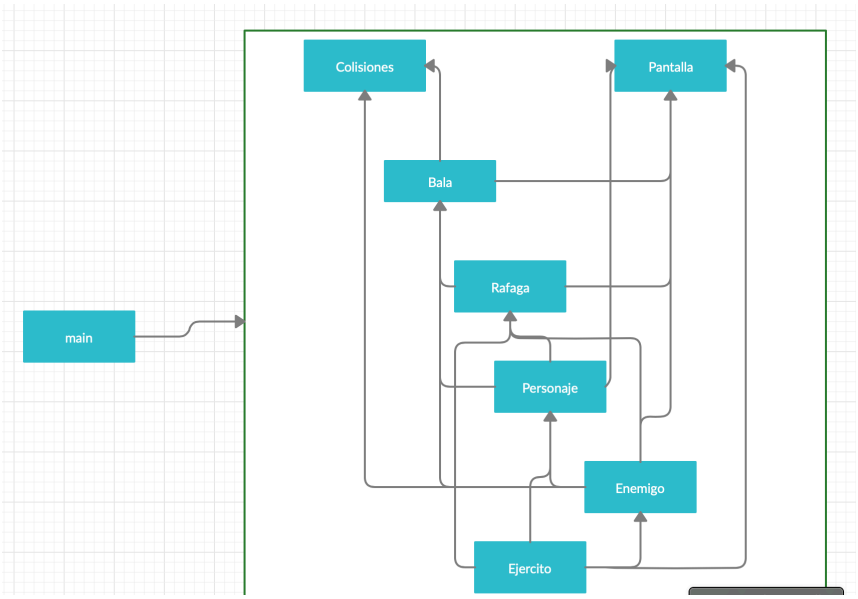

Figura 1. Diagrama de las relaciones y dependencias entre módulos. Fuente: propia.

3.2 MEJORAS INCLUIDAS

- #MásTDAs: se han creado y usado los TDA Personaje y Enemigo.
- #EjercitoDinamico: el array de enemigos es un array de dimensión variable, cuyo tamaño depende del parámetro que se pase como máximo al crear el ejército.
- #MasEnemigos: se ha añadido otro tipo de enemigo con movimiento distinto (perseguir al personaje) y representación diferente al del enemigo principal. También se ha añadido un "enemigo bueno" que otorga una vida al personaje.
- #Invulnerabilidad: el personaje es invulnerable durante un tiempo determinado tras perder una vida. Para ello, se ha incluido un campo en la estructura del personaje que indica si es invulnerable y un contador para llevar la cuenta del tiempo.
- #Animaciones: se ha conseguido que el personaje parpadee mientras sea invulnerable tras perder una vida.
- #ImagenCompartida: los enemigos comparten la imagen, no tiene cada uno su propia imagen en memoria. Para ello, la imagen de cada tipo de enemigo se ha incluido en la estructura del ejército.
- #EnemigosDisparan: los enemigos disparan balas con representación distinta a las del personaje.

4. ESTRUCTURAS DE DATOS

A continuación, se describen las estructuras de datos y TDA diseñados e implementados para el proyecto:

- Bala: la estructura de la bala está formada por sus coordenadas (xbala,ybala), su velocidad en el eje X "vxbala", su velocidad en el eje Y "vybala", su anchura "anchobala", su altura "altobala" y su imagen para ser representada gráficamente "imagenbala":

```
struct BalaRep
{
	double xbala;
	double ybala;
	double vxbala;
	double vybala;
	double anchobala;
	double altobala;
	Imagen imagenbala;
};
```

- Ejercito: implementado con representación contigua. La estructura está formada por el número de enemigos en un instante dado "NumEnemigos", el máximo de enemigos que se pueden almacenar "MaxEnemigos", un puntero a Enemigo "ejercito", y tres imágenes (una para cada tipo de enemigo posible): "imagenCoronavirus", "imagenSeta" y "imagenVacuna":

```
struct EjercitoRep
{
	int NumEnemigos;
	int MaxEnemigos;
	Enemigo * ejercito;
	Imagen imagenCoronavirus;
	Imagen imagenSeta;
	Imagen imagenVacuna;
};
```

- Enemigo: la estructura del enemigo está formada por sus coordenadas (x,y), su velocidad en el eje X “vx”, su velocidad en el eje Y “vy”, su anchura “ancho”, su altura “alto”, su numero de vidas “vidas”, su tipo y la imagen para ser representado gráficamente:

```
struct EnemigoRep
{
	double x;
	double y;
	double vx;
	double vy;
	double ancho;
	double alto;
	int vidas;
	Imagen imagen;
	int tipo;
};
```

- Personaje: la estructura del personaje está formada por sus coordenadas (x,y), su anchura “w”, su altura “h”, su velocidad en el eje X “vx”, su velocidad en el eje Y “vy”, su numero de vidas “vidas”, su numero de puntos “puntos”, un contador de invulnerabilidad “contInvulnerabilidad”, su “estado” de invulnerabilidad “invulnerabilidad” y una imagen para ser representado:

```
struct PersonajeRep
{
	double x;
	double y;
	double w;
	double h;
	double vx;
	double vy;
	int vidas;
	int puntos;
	Imagen imagen;
```

```
        int contInvulnerabilidad;
        int invulnerable;
    };
```

- Rafaga: implementado con representación enlazada. Su estructura está formada por una bala “BalaNodo” y una variable tipo Rafaga, “sig”, que apunta a la siguiente bala:

```
Struct RafagaRep
{
        Bala BalaNodo;
        Rafaga sig;
};
```

5. CONCLUSIONES

Personalmente, me ha parecido muy interesante y enriquecedora la experiencia de realizar este proyecto. Este juego ha sido mi primer proyecto de programación, tanto en C como en cualquier otro lenguaje, y he podido aprender, no solo el lenguaje, sino que también cómo solucionar y encontrar errores o mejorar el código para que un determinado procedimiento sea más eficiente, entre otros.

Obviamente, me he encontrado con ciertas dificultades, pero al final dedicándole tiempo al proyecto, y en algunos casos preguntando al profesor, he podido solucionarlas y el proyecto compila y se ejecuta sin ningún problema. No obstante, me gustaría comentar una pequeña dificultad que al final he solucionado de una forma no del todo satisfactoria. En el juego, quería que un determinado número de virus (el cual va aumentando) disparara, pero que fuese aleatorio, que no siempre dispararan los mismos. Lo implementé primero con rand, pero como se tenía que buscar un número aleatorio y comprobar que esa posición del ejército era un virus, se notaba que el programa a veces se ralentizaba mínimamente al disparar los enemigos. Además, nada aseguraba que un virus no disparara dos balas, que además saldrían superpuestas. Al final, decidí que lo mejor era que disparasen los primeros virus del ejército (el número dependerá de cuantos se quiera que disparen). Sin embargo, me hubiese gustado poder solucionar esto para que se ejecutase como se pensó en un principio, pero por falta de tiempo se optó por esta solución más sencilla.

También me hubiese gustado poder añadir más funcionalidades al juego, no solo de las que se incluyen en el boletín, sino también de invención propia. No obstante, al final considero que he podido implementar un buen número de mejoras de las que se muestran en el boletín. En concreto, he implementado: #MasTDAs, #EjercitoDinamico, #MasEnemigos, #Invulnerabilidad, #Animaciones, #ImagenCompartida y #EnemigosDisparan.

Para terminar, me gustaría comentar que la experiencia de trabajar en este proyecto haciendo uso de Code::Blocks ha sido satisfactoria, gracias a la gran cantidad de funcionalidades que otorga este entorno de desarrollo integrado. Además, este proyecto ha sido en gran parte responsable de mi alto grado de satisfacción con la asignatura, pues

aunque también ha influido la calidad de las clases teóricas y la implicación del profesor, poder ver que realmente se pone en práctica lo aprendido en estas clases es sumamente satisfactorio.