Práctica 2

ORB ATTACK

Francisco Méndez Vilas
3º Ingeniería Técnica Telecomunicaciones, esp. Telemática
Enero 2014

Índice

Memoria

Introducción

Orb Attack es un famoso juego de la era de los 90. Es un juego que consiste en una bola protagonista (bola verde) controlada por el usuario que tiene que tratar de permanecer el máximo tiempo en pantalla esquivando unas bolas rojas que le restan puntos si colisionan. En ayuda del usuario aparecen ocasionalmente bolas negras que destruyen las bolas rojas si colisionan y bolas azules que, mientras estén en pantalla, permiten al usuario disparar a las bolas rojas.

Estructura de la aplicación

Para la implementación de este juego se ha utilizado un proceso para cada bola y un proceso principal que es el encargado de controlar y coordinar los procesos, dibujar en pantalla y reproducir los sonidos.

Como función extra se ha implementado un menú inicial en el que opcionalmente se pueden mostrar las instrucciones del juego. Todo esto ejecutado en el proceso principal.

Una vez pasado el menú principal, el juego carga la configuración desde el fichero *config.txt* que debe estar estructurado de forma correcta. Con estos valores, se inicializa el juego y comienza la partida.

Comunicación entre procesos

Para la comunicación entre procesos se han utilizado 3 métodos: memoria compartida, tuberías y señales para coordinar la lectura/escritura de tuberías y creación/destrucción de bolas pequeñas.

Memoria compartida

- Juego: consiste en un array de 3 elementos: puntos (*long int*), fin de juego (*bool*), juego pausado (*bool*).
- Bola Verde: consiste en un array de 2 elementos: x e y.
- Bola Roja: consiste en un array de 4 elementos: x, y, radio, activa (*bool*). El tamaño de la memoria aquí es del tamaño de este array multiplicado por el número máximo de bolas rojas que puede haber en el juego.
- Bola Pequeña: consiste en un array de 2 elementos: x e y. El tamaño de la memoria aquí es del tamaño de este array multiplicado por el número máximo de bolas pequeñas que puede haber en el juego.
- Bolas Negras: consiste en un array de 3 elementos: x, y y activa (*bool*).

Tuberías

- Principal (tecla pulsada, `pipeTeclaBV`) → Bola Verde: Se utiliza para informar al proceso Bola Verde de la tecla que ha sido pulsada.

- Principal (es visible bola azul, `pipeIsVisibleBA`) → Bola Verde: Se utiliza para informar al proceso Bola Verde si la Bola Azul es visible.
- Bola Azul (posición, `pipePosBA`) → Principal: Se utiliza para que la Bola Azul informe de su posición al proceso principal.

Señales

- SIGTECLA: Utilizada para informar al proceso Bola Verde de que una tecla ha sido pulsada en el proceso principal y debe leer de la tubería asociada (`pipeTeclaBV`).
- SIGLEERAZUL: Utilizada para informar al proceso principal de que vamos a leer la información de la tubería asociada a la visibilidad de la Bola Azul (`pipeIsVisibleBA`).
- SIGMOVAZUL: Utilizada para informar al proceso principal de que la posición de la Bola Azul ha cambiado y que debe leer la posición de la tubería asociada (`pipePosBA`).
- SIGTOGGLEAZUL: Utilizada para indicar al proceso principal que debe cambiar la visibilidad de la Bola Azul. Si antes no estaba en pantalla ahora debe estarlo y viceversa.
- SIGCOMPEQUENA: Utilizada para indicar al proceso principal que debe crear una nueva Bola Pequeña.
- SIGDESTRUIRPEQUENA: Utilizada para indicar al proceso principal que debe destruir el proceso Bola Pequeña que le está emitiendo esta misma señal.

Manual de Usuario

Cómo empezar

El juego puede ser ejecutado desde consola o desde un explorar de archivos.

Consola:

```
$ ./p2
```

Explorador de archivos:

Ir a la carpeta que contiene el archivo ejecutable y hacer doble click en él.

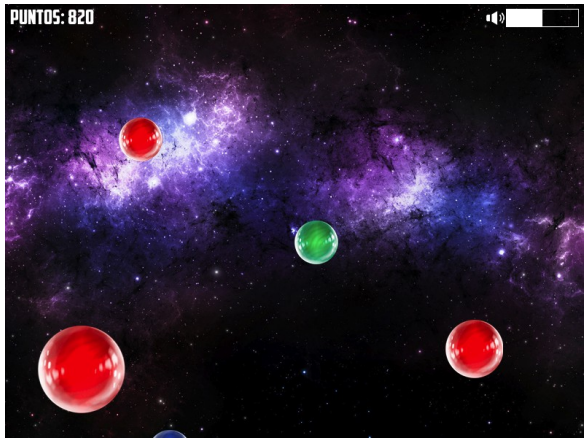
Pantalla de inicio

En la pantalla de inicio se puede pulsar la tecla 'I' para mostrar las instrucciones o ESC para salir del juego.

Jugando

Este juego consiste básicamente en que las bolas rojas no te toquen. Para esquivarlas puedes usar las flechas del teclado y desplazarte por toda la pantalla. Si disparas a las bolas rojas o las bolas negras colisionan con las rojas ganarás puntos. Estas son las teclas y sus funciones:

- Flechas: desplazamiento de la bola verde.
- Enter: disparar a las bolas rojas (sólo cuando las bolas azules estén en pantalla).
- Tecla '+': subir el volumen de la música.
- Tecla '-': bajar el volumen de la música.
- Tecla 'M': apagar/encender la música.
- Tecla 'P': pausar/reanudar el juego.
- ESC: salir del juego.



Configurar el juego

Todos los parámetros del juego pueden modificarse en el archivo config.txt. Dichos parámetros se detallan a continuación:

- energia_bolas_verdes: Los puntos iniciales con los que contarás.
- num_bolas_rojas: El número máximo de bolas rojas que puede haber en pantalla al mismo tiempo.
- intervalo_rojas: A cada cuantos segundos debe aparecer una nueva bola roja.
- intervalo_azules: A cada cuantos segundos debe aparecer la bola azul.
- num_bolas_negras: Número máximo de bolas negras que puede haber en pantalla al mismo tiempo.
- intervalo_negras: A cada cuantos segundos debe aparecer una nueva bola negra.
- bolas_verdes_pequeñas: Número máximo de disparos que puede haber al mismo tiempo en pantalla.
- puntuacion_t1_roja: Puntuación que ganarás si destruyes una bola roja de tamaño pequeño.
- puntuacion_t2_roja: Puntuación que ganarás si destruyes una bola roja de tamaño pequeño-mediano.
- puntuacion_t3_roja: Puntuación que ganarás si destruyes una bola roja de tamaño mediano.
- puntuacion_t4_roja: Puntuación que ganarás si destruyes una bola roja de tamaño grande.
- puntuacion_bola_negra: Puntuación que ganarás si una bola negra destruye una bola roja.
- quita_energia_roja: Puntos que se restarán si una bola roja logra alcanzarte.

Código con comentarios

P2.h

```
using namespace std;

struct Point {
	int   x;
	int   y;
};

struct PaqueteBP {
	char  orden[100];
	pid_t       pid = 0;
	Point pos = (Point){-1,-1};
	char  direccion = 0;
};

struct BolaPequena {
	pid_t             pid;
	Point             pos;
	char              direccion;
};

struct BolaRoja {
	int                     id;
	Point             pos;
	int                     radio;
	char              direccion;
};

struct BolaNegra {
	int                     id;
	Point             pos;
	char              direccion;
};

#define PI                          3.14159265
#define SIGTECLA              54
#define SIGMOVAZUL                  SIGUSR1
#define SIGTOGGLEAZUL         SIGUSR2
#define SIGLEERAZUL                 52
#define SIGCOMPEQUENA         51
#define SIGDESTRUIRPEQUENA    50
#define SIGPRINCIPALPEQUENA   49

// Gráficos
#define     FRAMES_PER_SECOND 200
#define     PANTALLA_ANCHO          800
#define     PANTALLA_ALTO           600

// Teclado
#define KEYS_PER_SECOND       50

// Juego
#define JUEGO_FINAL                 0
#define JUEGO_PAUSADO         1
#define JUEGO_PUNTOS          2
```

```
// Bola Verde
#define     BV_X                    0
#define     BV_Y                    sizeof(int)
#define BV_RADIO              30
#define PUNTOS_PERMANENCIA    10

// Bola Pequeña
#define BP_X                        0
#define BP_Y                        sizeof(int)
#define BP_RADIO              14

// Bola Roja
#define TIEMPO_CRECIMIENTO    3
#define     BR_X                    0
#define     BR_Y                    sizeof(int)
#define     BR_RADIO                sizeof(int) * 2
#define BR_ACTIVA             sizeof(int) * 3
#define     BR_RADIO_INICIAL  30
#define     BR_RADIO_T2             40
#define     BR_RADIO_T3             50
#define     BR_RADIO_T4             60
#define     BR_DIR_IZQ_ARRIBA 0
#define     BR_DIR_IZQ_ABAJO  1
#define     BR_DIR_DCHA_ARRIBA      2
#define     BR_DIR_DCHA_ABAJO 3
#define BR_VELOCIDAD          3
#define BR_POS_1              0
#define BR_POS_2              80
#define BR_POS_3              170
#define BR_POS_4              280

// Bola Azul
#define BA_X                        0
#define BA_Y                        sizeof(int)
#define BA_ACTIVA             sizeof(int) * 2
#define BA_RADIO              30

//Bola Negra
#define BN_X                        0
#define BN_Y                        sizeof(int)
#define BN_ACTIVA             sizeof(int) * 2
#define BN_RADIO              30
#define     BN_DIR_IZQ_ARRIBA 0
#define     BN_DIR_IZQ_ABAJO  1
#define     BN_DIR_DCHA_ARRIBA      2
#define     BN_DIR_DCHA_ABAJO 3
#define BN_VELOCIDAD          1

// Parametros del archivo
int         ENERGIA_BOLA_VERDE,
            NUM_BOLAS_ROJAS,
            INTERVALO_ROJAS,
            INTERVALO_AZUL,
            NUM_BOLAS_NEGRAS,
            INTERVALO_NEGRAS,
            NUM_BOLAS_PEQUENAS,
            PUNTUACION_T1_ROJA,
            PUNTUACION_T2_ROJA,
            PUNTUACION_T3_ROJA,
```

```
            PUNTUACION_T4_ROJA,
            PUNTUACION_NEGRA,
            QUITA_ENERGIA_ROJA;

int   pidPrincipal,
            pidBolaVerde,
            pidBolaRoja,
            pidBolaPequena,
            pidBolaAzul,
            pidBolaNegra;

long int    shmIdJuego;
int               *shmJuego;

long int    shmIdBolaVerde;
int               *shmBolaVerde;
char        tecla;
Point       posBV;
int         pipeTeclaBV[2];

long int    shmIdBolaRoja;
int               *shmBolaRoja;
int         shmTamanoBolaRoja;
SDL_TimerID *timerIdRoja;

BolaPequena bolasPequenas[5];
BolaPequena bp;
int         bpUltimoId;
int         bpTotal;
long int    shmIdBolaPequena;
int               *shmBolaPequena;
int         shmTamanoBolaPequena;

int         pipePosBA[2];
int         pipeIsVisibleBA[2];
Point       posAzul;
bool        isVisibleAzul;

long int    shmIdBolaNegra;
int               *shmBolaNegra;
int         shmTamanoBolaNegra;
SDL_TimerID *timerIdNegra;

SDL_Surface       *pantalla,
                        *srfInicio,
                        *srfInstrucciones,
                        *fondo,
                        *srfFlash,
                        *srfEnPausa,
                        *srfHasPerdido,
                        *srfBolaVerde,
                        *srfBolaRoja,
                        *srfBolaPequena,
                        *srfBolaAzul,
                        *srfBolaNegra,
                        *srfPuntuacion,
                        *srfVolumen;

TTF_Font          *fontPuntuacion;
SDL_Color         colorBlanco,
                        colorRojo;
```

```
Mix_Music         *musica,
                        *musicaInicio;

Mix_Chunk         *shot,
                        *explosion,
                        *whip;

int main();
void procesoPrincipal();
void registerSigMovAzul();
void registerSigToggleAzul();
void registerSigLeerAzul();
void registerSigComPequena();
void registerSigDestruirPequena();
int crearProcesoBolaVerde();
void gestionarTeclaBolaVerde(int);
void dibujarBolaVerde(int, int);
int crearProcesoBolaPequena(int, char);
void sigComPequena(int);
void sigDestruirPequena(int, siginfo_t*, void*);
int obtenerNuevoIdPequena();
void registerSigPrincipalPequena();
void dibujarBolaPequena();
int crearProcesoBolaRoja();
int getIdLibreRoja();
void inicializarBolaRoja(int, BolaRoja*);
Uint32 moverBolaRoja(Uint32, void*);
void dibujarBolaRoja();
int crearProcesoBolaAzul();
void sigBA(int);
void sigToggleBA(int);
void sigLeerBA(int);
void dibujarBolaAzul();
int crearProcesoBolaNegra();
int getIdLibreNegra();
void inicializarBolaNegra(int, BolaNegra*);
Uint32 moverBolaNegra(Uint32, void*);
void dibujarBolaNegra();
void inicializar();
void cargarConfiguracion();
void mostrarPantallaInicio();
void dibujarFondo();
void dibujarFlash();
void dibujarEnPausa();
void dibujarHasPerdido();
void dibujarPuntuacion();
void dibujarVolumen();
int bolaRojaMasCercana();
bool detectarColisionBVBR();
double distanciaBolaVerdeBolaRoja(int);
void detectarColisionBPBR();
double distanciaBolaPequenaBolaRoja(int, int);
void detectarColisionBPBA();
int detectarColisionBNBR();
double distanciaBolaNegraBolaRoja(int, int);
void liberar();
int* crearVariosProcesos();
```

P2.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <signal.h>
#include <string.h>
#include <errno.h>
#include <time.h>
#include <cmath>
#include <fstream>
#include <sstream>
#include <sys/types.h>
#include <sys/wait.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/time.h>
#include <SDL.h>
#include <SDL_image.h>
#include <SDL_mixer.h>
#include <SDL_ttf.h>

#include "p2.h"

int main() {
	mostrarPantallaInicio();
	cargarConfiguracion();

	// MEMORIA COMPARTIDA ////////////////////
	// sizeof(long int)+2: puntos (long int) + fin de juego (bool->char) +
juego pausado (bool->char)
	shmIdJuego = shmget( (key_t)ftok("/bin/ls", 11), sizeof(long int) + 2,
0777 | IPC_CREAT );
	shmJuego = (int *)shmat(shmIdJuego, NULL, 0);
	memset(shmJuego, 0, sizeof(long int) + 2);

	// sizeof(int)*2: x (int) + y (int)
	shmIdBolaVerde = shmget( (key_t)ftok("/bin/ls", 22), sizeof(int)*2, 0777 |
IPC_CREAT );
	shmBolaVerde = (int *)shmat(shmIdBolaVerde, NULL, 0);
	memset(shmBolaVerde, 0, sizeof(int)*2);

	// (sizeof(int)*3+1)*NUM_BOLAS_ROJAS: x (int) + y (int) + radio (int) +
activa (bool->char)
	shmTamanoBolaRoja = sizeof(int) * 3 + 1;
	shmIdBolaRoja = shmget( (key_t)ftok("/bin/ls", 33), shmTamanoBolaRoja *
NUM_BOLAS_ROJAS, 0777 | IPC_CREAT );
	shmBolaRoja = (int *)shmat(shmIdBolaRoja, NULL, 0);
	memset(shmBolaRoja, 0, shmTamanoBolaRoja);

	// (sizeof(int)*2)*NUM_BOLAS_PEQUENAS: (x (int) + y (int)) *
NUM_BOLAS_PEQUENAS
	shmTamanoBolaPequena = sizeof(int) * 2 * NUM_BOLAS_PEQUENAS;
	shmIdBolaPequena = shmget( (key_t)ftok("/bin/ls", 33),
shmTamanoBolaPequena, 0777 | IPC_CREAT );
	shmBolaPequena = (int *)shmat(shmIdBolaPequena, NULL, 0);
	memset(shmBolaPequena, 0, shmTamanoBolaPequena);

	// (sizeof(int)*2+1)*NUM_BOLAS_NEGRAS: x (int) + y (int) + activa (bool-
>char)
```

```
	shmTamanoBolaNegra = sizeof(int) * 2 + 1;
	shmIdBolaNegra = shmget( (key_t)ftok("/bin/ls", 66), shmTamanoBolaNegra *
NUM_BOLAS_NEGRAS, 0777 | IPC_CREAT );
	shmBolaNegra = (int *)shmat(shmIdBolaNegra, NULL, 0);
	memset(shmBolaNegra, 0, shmTamanoBolaNegra);

	// Registrar señales de la bola azul
	registerSigMovAzul();
	registerSigToggleAzul();
	registerSigLeerAzul();

	// Registrar señales de las bolas pequeñas
	registerSigComPequena();
	registerSigDestruirPequena();

	pidPrincipal = getpid();

	pidBolaVerde = crearProcesoBolaVerde();

	if( pidBolaVerde > 0 ) {
		pidBolaRoja = crearProcesoBolaRoja();

		if( pidBolaRoja > 0 ) {
			pidBolaNegra = crearProcesoBolaNegra();

			if( pidBolaNegra > 0 ) {
				procesoPrincipal();
			}
		}
	}

	// Aseguramos que siempre se cierren todos los procesos
	kill(pidBolaNegra, SIGKILL);
	kill(pidBolaAzul, SIGKILL);
	kill(pidBolaRoja, SIGKILL);
	kill(pidBolaVerde, SIGKILL);

	return 0;
}

void procesoPrincipal() {
	int		frame = 0,
			idRoja;
	SDL_Event	event;
	Uint32		fpsStart,
			contadorAzul = SDL_GetTicks(),
			retardoTeclaMusica = SDL_GetTicks(),
			retardoTeclaPausa = SDL_GetTicks(),
			retardoTeclaEnter = SDL_GetTicks();

	// Inicializamos graficos, sonido y variables del juego
	inicializar();

	// Inicializamos las variables que controlan las bolas
	// pequeñas y la azul.
	isVisibleAzul = false;
	for(int i=0; i < NUM_BOLAS_PEQUENAS; i++) {
		bolasPequenas[i].pid = 0;
	}
	bpUltimoId = 0;
	bpTotal = 0;
```

```
	// Bucle de ejecución principal
	do {

		// Recogemos el evento (sin bloquear la ejecucion) y
		// aplicamos los cambios segun el tipo de evento.
		SDL_PollEvent(&event);

		if( event.type == SDL_KEYDOWN ) {
			tecla = event.key.keysym.sym;

			// Si la tecla es 'p' pausamos/reanudamos el juego
			if( event.key.keysym.sym == SDLK_p ) {
				if( SDL_GetTicks() - retardoTeclaPausa >= 800 ) {
					if( shmJuego[JUEGO_PAUSADO] == 0 ) {
						shmJuego[JUEGO_PAUSADO] = 1;
						Mix_PauseMusic();
						dibujarEnPausa();
						SDL_Flip(pantalla);
					} else {
						shmJuego[JUEGO_PAUSADO] = 0;
						Mix_ResumeMusic();
					}

					retardoTeclaPausa = SDL_GetTicks();
				}
			// Si la tecla es 'm' apagamos/encendemos la musica
			} else if( event.key.keysym.sym == SDLK_m ) {
				if( Mix_PlayingMusic() && SDL_GetTicks() -
retardoTeclaMusica >= 800 ) {
					if( Mix_PausedMusic() ) {
						Mix_ResumeMusic();
					} else {
						Mix_PauseMusic();
					}

					retardoTeclaMusica = SDL_GetTicks();
				}
			// Si la tecla es Enter notificamos a Bola Verde que debe
disparar
			} else if( event.key.keysym.sym == SDLK_RETURN ) {
				if( SDL_GetTicks() - retardoTeclaEnter >= 300 ) {
					kill(pidBolaVerde, SIGTECLA);
					SDL_Delay(1);
					write(pipeTeclaBV[1], &tecla, sizeof(char));
					retardoTeclaEnter = SDL_GetTicks();
				}
			}

			// Esto eliminará el mal efecto creado por la rapida
			// repeticion de teclas.
			if( frame % KEYS_PER_SECOND == 0 ) {
				switch(event.key.keysym.sym) {
					case SDLK_PLUS: // Subir el volumen
						if( Mix_VolumeMusic(-1) != MIX_MAX_VOLUME )
{
							Mix_VolumeMusic(Mix_VolumeMusic(-1) +
1);
							Mix_Volume(-1, Mix_VolumeMusic(-1) +
1);
						}
```

```
                              break;
                              case SDLK_MINUS: // Bajar el volumen
                                    if( Mix_VolumeMusic(-1) != 0 ) {
                                          Mix_VolumeMusic(Mix_VolumeMusic(-1) -
1);
                                          Mix_Volume(-1, Mix_VolumeMusic(-1) -
1);
                                    }
                              break;
                              case SDLK_RETURN:
                                     // Evitamos que cuando se pulse Enter
                                     // se vuelva a notificar a Bola Verde
                              break;
                              default: // Notificamos a Bola Verde de la tecla
pulsada
                                    kill(pidBolaVerde, SIGTECLA);
                                    SDL_Delay(1);
                                    write(pipeTeclaBV[1], &tecla,
sizeof(char));
                              break;
                        }
                  }
            // Si se cierra la ventana con el raton pulsando en el boton X de
la
            // esquina o mediante Alt+F4 o similar.
            } else if( event.type == SDL_QUIT ) {
                  shmJuego[JUEGO_FINAL] = 1;
            }


            // Si el juego no esta pausado...
            if( !shmJuego[JUEGO_PAUSADO] ) {

                  // Dibujamos la pantalla a una velocidad de FRAMES_PER_SECOND
                  // para evitar que se hagan demasiados calculos por segundo.
                  if( frame % FRAMES_PER_SECOND == 0 ) {
                        dibujarFondo();

                        // Detectar colision Bola Pequeña - Bola Roja
                        detectarColisionBPBR();

                        // Detectar colision Bola Negra - Bola Roja
                        idRoja = detectarColisionBNBR();
                        if( idRoja != -1 ) {
                              shmBolaRoja[BR_ACTIVA + idRoja *
shmTamanoBolaRoja] = 0;
                        }

                        // Detectar colision Bola Pequeña - Bola Azul
                        detectarColisionBPBA();

                        // Detectar colision Bola Verde - Bola Roja
                        if( detectarColisionBVBR() ) {
                              dibujarFlash();
                        }

                        // Dibujar las bolas
                        if( isVisibleAzul ) dibujarBolaAzul();
                        dibujarBolaVerde(shmBolaVerde[BV_X],
shmBolaVerde[BV_Y]);
                        dibujarBolaRoja();
```

```
                        dibujarBolaNegra();
                        dibujarBolaPequena();

                        // Si los puntos son negativos se ponen a cero y se
                        // termina el juego.
                        if( shmJuego[JUEGO_PUNTOS] <= 0 ) {
                              shmJuego[JUEGO_PUNTOS] = 0;
                              shmJuego[JUEGO_FINAL] = 1;
                        }

                        // Dibujar los datos superiores de la pantalla
                        dibujarPuntuacion();
                        dibujarVolumen();

                        // Renderizar la pantalla
                        SDL_Flip(pantalla);
                        frame = 0;
                  }

                  // Actualizar la tasa de frames por segundo
                  if( SDL_GetTicks() - fpsStart < 1000 ) {
                  frame++;
              } else {
                  fpsStart = SDL_GetTicks();
              }

            // Actualizar el contador de tiempo para la Bola Azul y
              // si ha pasado suficiente tiempo, volver a mostrar.
            if( !isVisibleAzul ) {
                  if( SDL_GetTicks() - contadorAzul >=
(Uint32)INTERVALO_AZUL*1000 ) {
                  contadorAzul = SDL_GetTicks();
                  pidBolaAzul = crearProcesoBolaAzul();
                  }
            } else {
                  contadorAzul = SDL_GetTicks();
            }

          }

      // Permanecer en el bucle a menos que se pulse ESC o JUEGO_FINAL diferente
de 0.
      } while(event.key.keysym.sym != SDLK_ESCAPE && shmJuego[JUEGO_FINAL] ==
0);

      // Se salga como se salga, ponemos JUEGO_FINAL a 1 para que el resto de
      // procesos sepan que todo ha terminado.
      shmJuego[JUEGO_FINAL] = 1;

      if( shmJuego[JUEGO_PUNTOS] <= 0 ) { // Si se pierde la partida
            Mix_HaltMusic();
            dibujarHasPerdido();
            SDL_Flip(pantalla);
            SDL_Delay(4000);
            SDL_WaitEvent(NULL);
      } else { // Si se sale voluntariamente
            Mix_FadeOutMusic(1000);
            SDL_Delay(1000);
      }
}
```

```
void registerSigMovAzul() {
	// Manejador de señal SIGMOVAZUL
	struct sigaction act;
	act.sa_handler = &sigBA;
	act.sa_flags = SA_RESTART;
	sigemptyset(&act.sa_mask);
	sigaction(SIGMOVAZUL, &act, NULL);
}

void registerSigToggleAzul() {
	// Manejador de señal SIGTOGGLEAZUL
	struct sigaction act;
	act.sa_handler = &sigToggleBA;
	act.sa_flags = SA_RESTART;
	sigemptyset(&act.sa_mask);
	sigaction(SIGTOGGLEAZUL, &act, NULL);
}

void registerSigLeerAzul() {
	// Manejador de señal SIGLEERAZUL
	struct sigaction act;
	act.sa_handler = &sigLeerBA;
	act.sa_flags = SA_RESTART;
	sigemptyset(&act.sa_mask);
	sigaction(SIGLEERAZUL, &act, NULL);
}

void registerSigComPequena() {
	// Manejador de señal SIGCOMPEQUENA
	struct sigaction act;
	act.sa_handler = &sigComPequena;
	act.sa_flags = SA_RESTART;
	sigemptyset(&act.sa_mask);
	sigaction(SIGCOMPEQUENA, &act, NULL);
}

void registerSigDestruirPequena() {
	// Manejador de señal SIGDESTRUIRPEQUENA
	struct sigaction act;
	act.sa_sigaction = &sigDestruirPequena;
	act.sa_flags = SA_RESTART | SA_SIGINFO;
	sigemptyset(&act.sa_mask);
	sigaction(SIGDESTRUIRPEQUENA, &act, NULL);
}

int crearProcesoBolaVerde() {
	pid_t pid;

	// Canal de comunicacion para que la Bola Verde
	// reciba que tecla se ha pulsado.
	if( pipe(pipeTeclaBV) == -1 ) {
		printf("Error al crear pipe pipeTeclaBV.\n");
	}

	// Canal de comunicacion para que la Bola Verde
	// reciba si la Bola Azul esta visible o no.
	if( pipe(pipeIsVisibleBA) == -1 ) {
		printf("Error al crear pipe pipeIsVisibleBA.\n");
	}

	pid = fork();
```

```
	if( pid == 0 ) { //Proceso hijo
		close(pipeTeclaBV[1]); // Cerrar salida pipe para hijo
		close(pipeIsVisibleBA[1]); // Cerrar salida pipe para hijo

		// Manejador de señal SIGTECLA
		struct sigaction act;
		act.sa_handler = &gestionarTeclaBolaVerde;
		act.sa_flags = SA_RESTART;
		sigemptyset(&act.sa_mask);
		sigaction(SIGTECLA, &act, NULL);

		Uint32 inicio = SDL_GetTicks();


		// Inicializando la posicion de la bola verde
		posBV.x = PANTALLA_ANCHO / 2;
		posBV.y = PANTALLA_ALTO / 2;
		shmBolaVerde[BV_X] = posBV.x;
		shmBolaVerde[BV_Y] = posBV.y;

		printf("Bola Verde iniciado...\n");
		do {
			if( !shmJuego[JUEGO_PAUSADO] ) {
				if( SDL_GetTicks() - inicio >= 1000 ) {
					shmJuego[JUEGO_PUNTOS] += PUNTOS_PERMANENCIA;
					inicio = SDL_GetTicks();
				}
			} else {
				inicio = SDL_GetTicks();
			}
		} while(!shmJuego[JUEGO_FINAL]);
		printf("Bola Verde finalizado...\n");
		exit(0);
	} else if( pid == -1 ) {
		perror("Error al crear el proceso Bola Verde.");
		exit(1);
	}

	close(pipeTeclaBV[0]); //Cerrar entrada pipe para padre
	close(pipeIsVisibleBA[0]); //Cerrar entrada pipe para padre

	return pid;
}

// Cuando se recibe la señal se lee del pipe
// pipeTeclaBV para saber cual fue la tecla que
// se ha pulsado y actuar en consecuencia.
void gestionarTeclaBolaVerde(int signal) {
	read(pipeTeclaBV[0], &tecla, sizeof(char));

	switch(tecla) {
		case (char)SDLK_LEFT:
			posBV.x = (posBV.x > BV_RADIO) ? posBV.x-10 : 0;
			shmBolaVerde[BV_X] = posBV.x;
		break;
		case (char)SDLK_RIGHT:
			posBV.x = (posBV.x < PANTALLA_ANCHO-BV_RADIO*2) ? posBV.x+10 :
PANTALLA_ANCHO-BV_RADIO*2;
			shmBolaVerde[BV_X] = posBV.x;
		break;
```

```
            case (char)SDLK_UP:
                  posBV.y = (posBV.y > BV_RADIO) ? posBV.y-10 : 0;
                  shmBolaVerde[BV_Y] = posBV.y;
            break;
            case (char)SDLK_DOWN:
                  posBV.y = (posBV.y < PANTALLA_ALTO-BV_RADIO*2) ? posBV.y+10 :
PANTALLA_ALTO-BV_RADIO*2;
                  shmBolaVerde[BV_Y] = posBV.y;
            break;
            case (char)SDLK_RETURN:
                  bool visible;

                  // Si Bola Azul esta visible crear Bola Pequeña
                  kill(pidPrincipal, SIGLEERAZUL);
                  read(pipeIsVisibleBA[0], &visible, 1);
                  if( visible ) {
                        kill(pidPrincipal, SIGCOMPEQUENA);
                  }
            break;
      }
}

void dibujarBolaVerde(int x, int y) {
      SDL_Rect rect;

      rect = (SDL_Rect) { (Sint16)x, (Sint16)y, BV_RADIO*2, BV_RADIO*2 };
      SDL_BlitSurface(srfBolaVerde, NULL, pantalla, &rect);
}

int crearProcesoBolaPequena(int id, char dir) {
      pid_t pid;

      pid = fork();

      if( pid == 0 ) { // Proceso hijo
            printf("Bola Pequena iniciado (%d)...\n", getpid());
            int aumento;
            int x;

            // Si la bola debe ir hacia la izquierda
            // el aumento sera de -1 pixel cada vez. En caso
            // contrario sera de 1 px.
            if( dir == 'I' ) {
                  aumento = -1;
            } else {
                  aumento = 1;
            }

            // Obtenemos la posicion x inicial de la Bola Pequeña
            // La coordenada 'y' habra sido inicializada en el proceso
            // Bola Verde, cuando se pulso Enter.
            x = shmBolaPequena[BP_X + id * shmTamanoBolaPequena];

            do {
                  while(shmJuego[JUEGO_PAUSADO] && !shmJuego[JUEGO_FINAL]);

                  x += aumento;
                  shmBolaPequena[BP_X + id * shmTamanoBolaPequena] = x;
                  SDL_Delay(1);
            } while(x >= -BP_RADIO*2 && x <= PANTALLA_ANCHO &&
shmJuego[JUEGO_FINAL] == 0);
```

```
            kill(pidPrincipal, SIGDESTRUIRPEQUENA);
            printf("Bola Pequena finalizado...\n");
            quick_exit(0);
      } else if( pid == -1 ) {
            perror("Error al crear el proceso Bolas Pequenas.");
            exit(1);
      }

      return pid;
}

void sigComPequena(int signal) {
      // Controlamos que no hayamos excedido el numero maximo
      // de Bolas Pequeñas en pantalla.
      if( bpTotal < NUM_BOLAS_PEQUENAS ) {
            pid_t pidBP;
            int id;

            id = obtenerNuevoIdPequena();

            if( id != -1 ) {
                  int idRoja = bolaRojaMasCercana();
                  char dir;

                  // En funcion de la bola roja mas cercana, la bola
                  // pequeña debera ir hacia la izquierda o la derecha.
                  if( idRoja != -1 ) {
                        if( shmBolaRoja[BR_X + idRoja * shmTamanoBolaRoja] <=
shmBolaVerde[BV_X] ) {
                              dir = 'I';
                        } else {
                              dir = 'D';
                        }
                  } else { // Si no habia bolas rojas en pantalla elegir
izquierda (por ejemplo)
                        dir = 'I';
                  }

                  // Inicializar posicion de la bola pequeña en funcion de la
verde.
                  shmBolaPequena[BP_X + id * shmTamanoBolaPequena] =
shmBolaVerde[BV_X] - BP_RADIO * 2;
                  shmBolaPequena[BP_Y + id * shmTamanoBolaPequena] =
shmBolaVerde[BV_Y] + BV_RADIO - BP_RADIO;

                  // Crear el proceso bola pequeña
                  pidBP = crearProcesoBolaPequena(id, dir);
                  bolasPequenas[id].pid = pidBP;
                  bpTotal++;
            }
      }
}

void sigDestruirPequena(int signal, siginfo_t *info, void *data) {
      bool encontrado = false;

      bpTotal--;

      for(int i=0; i < NUM_BOLAS_PEQUENAS && !encontrado; i++) {
            if( bolasPequenas[i].pid == info->si_pid ) {
                  bolasPequenas[i].pid = 0;
```

```
            }
      }
}


// Esta funcion busca en un array de tamaño NUM_BOLAS_PEQUENAS si hay
// algun hueco libre donde alojar una nueva Bola Pequeña. Es un algoritmo
// circular.
int obtenerNuevoIdPequena() {
      bpUltimoId++;

      // Si sobrepasa el limite, volver al comienzo del array
      if( bpUltimoId >= NUM_BOLAS_PEQUENAS ) {
            bpUltimoId = 0;
      }

      // Si ese lugar del array ya esta ocupado buscamos en todo
      // el array tratando de encontrar un hueco libre.
      if( bolasPequenas[bpUltimoId].pid ) {
            bool encontrado = false;

            for(int i=0; i < NUM_BOLAS_PEQUENAS && !encontrado; i++) {
                  bpUltimoId++;
                  // Como antes, si sobrepasa limite, volvemos a empezar.
                  if( bpUltimoId >= NUM_BOLAS_PEQUENAS ) bpUltimoId = 0;

                  // Si el hueco esta vacio actualizar el flag encontrado
                  // y salir del bucle
                  if( !bolasPequenas[bpUltimoId].pid ) {
                        encontrado = true;
                  }
            }

            // Si tras la busqueda no hay ningun hueco libre, devolver -1
            if( !encontrado ) return -1;
      }

      return bpUltimoId;
}

void registerSigPrincipalPequena() {
      // Manejador de señal SIGPRINCIPALPEQUENA
      struct sigaction act;
      act.sa_handler = &sigComPequena;
      act.sa_flags = SA_RESTART;
      sigemptyset(&act.sa_mask);
      sigaction(SIGPRINCIPALPEQUENA, &act, NULL);
}

void dibujarBolaPequena() {
      SDL_Rect rect;

      for(int i=0; i < NUM_BOLAS_PEQUENAS; i++) {
            if( bolasPequenas[i].pid ) {
                  rect = (SDL_Rect) { (Sint16)shmBolaPequena[BP_X +
i*shmTamanoBolaPequena], (Sint16)shmBolaPequena[BP_Y + i*shmTamanoBolaPequena],
BP_RADIO*2, BP_RADIO*2 };
                  SDL_BlitSurface(srfBolaPequena, NULL, pantalla, &rect);
            }
      }
}
```

```
// Este proceso controla las bolas rojas
int crearProcesoBolaRoja() {
	pid_t	pid;

	pid = fork();

	if( pid == 0 ) { //Proceso hijo
		Uint32			inicio, // para controlar la velocidad de
movimiento
					inicioCrecimiento[NUM_BOLAS_ROJAS]; // para
controlar lo que tardan en crecer
		BolaRoja		br[NUM_BOLAS_ROJAS]; // Array con todas las bolas
rojas
		int			nextId = 0; // id para ir iterando entre las
bolas

		// Inicializamos todas las bolas rojas a sus valores por defecto
		for(int i=0; i < NUM_BOLAS_ROJAS; i++) {
			inicioCrecimiento[i] = 0;
			timerIdRoja[i] = (SDL_TimerID)-1;
			inicializarBolaRoja(i, br);
		}

		// Inicializando timer
		if (SDL_Init(SDL_INIT_TIMER) == -1) {
			printf("No se pudo iniciar timer: %s\n", SDL_GetError());
			exit(1);
		}

		// Introducir un pequeño delay para que el juego no
		// comience inmediatamente y al usuario le de tiempo a
"prepararse".
		SDL_Delay(2000);

		printf("Bola Roja iniciado...\n");

		// Inicializamos timers y activamos la primera bola roja
		inicio = SDL_GetTicks();
		shmBolaRoja[BR_ACTIVA] = 1;
		timerIdRoja[0] = SDL_AddTimer(20, moverBolaRoja, &br[0]);
		inicioCrecimiento[0] = SDL_GetTicks();

		while(!shmJuego[JUEGO_FINAL]) {

			while(shmJuego[JUEGO_PAUSADO] && !shmJuego[JUEGO_FINAL]) {
				inicio = SDL_GetTicks();
				for(int i=0; i < NUM_BOLAS_ROJAS; i++) {
					inicioCrecimiento[i] = SDL_GetTicks();
				}
			}

			if( SDL_GetTicks() - inicio >= (Uint32)(INTERVALO_ROJAS *
1000) ) {
				// Si quedan bolas rojas libres, inicializarla y
sacarla.
				nextId = getIdLibreRoja();
				if( nextId != -1 ) {
					inicializarBolaRoja(nextId, br);
					shmBolaRoja[BR_ACTIVA + nextId*shmTamanoBolaRoja]
```

```
= 1;
                              timerIdRoja[nextId] = SDL_AddTimer(20,
moverBolaRoja, &br[nextId]);
                              inicioCrecimiento[nextId] = SDL_GetTicks();
                        }
                        inicio = SDL_GetTicks();
                  }

                  // Controlar el crecimiento de las bolas segun pasa el tiempo
                  for(int i=0; i < NUM_BOLAS_ROJAS; i++) {
                        if( inicioCrecimiento[i] != 0 && SDL_GetTicks() -
inicioCrecimiento[i] >= TIEMPO_CRECIMIENTO * 1000 ) {
                              if( br[i].radio == BR_RADIO_INICIAL ) {
                                    br[i].radio = BR_RADIO_T2;
                              } else if( br[i].radio == BR_RADIO_T2 ) {
                                    br[i].radio = BR_RADIO_T3;
                              } else if( br[i].radio == BR_RADIO_T3 ) {
                                    br[i].radio = BR_RADIO_T4;
                              }

                              shmBolaRoja[BR_RADIO + i*shmTamanoBolaRoja] =
br[i].radio;

                              inicioCrecimiento[i] = SDL_GetTicks();
                        }
                  }
            }

            // Eliminar todos los timers
            for(int i=0; i < NUM_BOLAS_ROJAS; i++) {
                  if( timerIdRoja[i] != (SDL_TimerID)-1 )
SDL_RemoveTimer(timerIdRoja[i]);
            }

            printf("Bola Roja finalizado...\n");
            exit(0);
      } else if( pid == -1 ) {
            perror("Error al crear el proceso Bola Roja.");
            exit(1);
      }

      return pid;
}

// Buscar una bola roja libre y retornar su id
int getIdLibreRoja() {
      bool encontrado = false;
      int id = -1;

      for(int i=0; i < NUM_BOLAS_ROJAS && !encontrado; i++) {
            if( timerIdRoja[i] == (SDL_TimerID)-1 ) {
                  id = i;
                  encontrado = true;
            }
      }

      return id;
}

void inicializarBolaRoja(int id, BolaRoja *br) {
      br[id].id = id;
```

```
	br[id].pos.x = -BR_RADIO_INICIAL*2;
	br[id].pos.y = -BR_RADIO_INICIAL*2;
	br[id].radio = BR_RADIO_INICIAL;
	br[id].direccion = BR_DIR_DCHA_ABAJO;
	shmBolaRoja[BR_X + id*shmTamanoBolaRoja] = br[id].pos.x;
	shmBolaRoja[BR_Y + id*shmTamanoBolaRoja] = br[id].pos.y;
	shmBolaRoja[BR_RADIO + id*shmTamanoBolaRoja] = BR_RADIO_INICIAL;
	shmBolaRoja[BR_ACTIVA + id*shmTamanoBolaRoja] = 0;
}

// Controlar el movimiento de las bolas rojas
Uint32 moverBolaRoja(Uint32 interval, void *param) {
	if( !shmJuego[JUEGO_PAUSADO] ) {
		BolaRoja *br = (BolaRoja *)param;
		int desplazamiento = br->id * shmTamanoBolaRoja;

		if( shmBolaRoja[BR_ACTIVA + desplazamiento] ) {

			switch(br->direccion) {
				case BR_DIR_IZQ_ARRIBA:
					br->pos.x = (br->pos.x > BR_VELOCIDAD) ? br-
>pos.x-BR_VELOCIDAD : 0;
					br->pos.y = (br->pos.y > BR_VELOCIDAD) ? br-
>pos.y-BR_VELOCIDAD : 0;

					// Limite esquina superior izquierda
					if( br->pos.x == 0 && br->pos.y == 0 ) {
						br->direccion = BR_DIR_DCHA_ABAJO;
					// Limite izquierdo
					} else if( br->pos.x == 0 && br->pos.y != 0 ) {
						br->direccion = BR_DIR_DCHA_ARRIBA;
					// Limite superior
					} else if( br->pos.x != 0 && br->pos.y == 0 ) {
						br->direccion = BR_DIR_IZQ_ABAJO;
					}
				break;
				case BR_DIR_IZQ_ABAJO:
					br->pos.x = (br->pos.x > BR_VELOCIDAD) ? br-
>pos.x-BR_VELOCIDAD : 0;
					br->pos.y = (br->pos.y < PANTALLA_ALTO-
BR_VELOCIDAD-br->radio*2) ? br->pos.y+BR_VELOCIDAD : PANTALLA_ALTO-br->radio*2;

					// Esquina inferior izquierda
					if( br->pos.x == 0 && br->pos.y == PANTALLA_ALTO-
br->radio*2 ) {
						br->direccion = BR_DIR_DCHA_ARRIBA;
					// Limite izquierdo
					} else if( br->pos.x == 0 && br->pos.y !=
PANTALLA_ALTO-br->radio*2 ) {
						br->direccion = BR_DIR_DCHA_ABAJO;
					// Limite inferior
					} else if( br->pos.x != 0 && br->pos.y ==
PANTALLA_ALTO-br->radio*2 ) {
						br->direccion = BR_DIR_IZQ_ARRIBA;
					}
				break;
				case BR_DIR_DCHA_ARRIBA:
					br->pos.x = (br->pos.x < PANTALLA_ANCHO-
BR_VELOCIDAD-br->radio*2) ? br->pos.x+BR_VELOCIDAD : PANTALLA_ANCHO-br-
>radio*2;
					br->pos.y = (br->pos.y > BR_VELOCIDAD) ? br-
```

```
>pos.y-BR_VELOCIDAD : 0;

                              // Esquina superior derecha
                              if( br->pos.x == PANTALLA_ANCHO-br->radio*2 && br-
>pos.y == 0 ) {
                                    br->direccion = BR_DIR_IZQ_ABAJO;
                              // Limite derecho
                              } else if( br->pos.x == PANTALLA_ANCHO-br->radio*2
&& br->pos.y != 0 ) {
                                    br->direccion = BR_DIR_IZQ_ARRIBA;
                              // Limite superior
                              } else if( br->pos.x != PANTALLA_ANCHO-br->radio*2
&& br->pos.y == 0 ) {
                                    br->direccion = BR_DIR_DCHA_ABAJO;
                              }
                        break;
                        case BR_DIR_DCHA_ABAJO:
                              br->pos.x = (br->pos.x < PANTALLA_ANCHO-
BR_VELOCIDAD-br->radio*2) ? br->pos.x+BR_VELOCIDAD : PANTALLA_ANCHO-br-
>radio*2;
                              br->pos.y = (br->pos.y < PANTALLA_ALTO-
BR_VELOCIDAD-br->radio*2) ? br->pos.y+BR_VELOCIDAD : PANTALLA_ALTO-br->radio*2;

                              // Esquina inferior derecha
                              if( br->pos.x == PANTALLA_ANCHO-br->radio*2 && br-
>pos.y == PANTALLA_ALTO-br->radio*2 ) {
                                    br->direccion = BR_DIR_IZQ_ARRIBA;
                              // Limite derecho
                              } else if( br->pos.x == PANTALLA_ANCHO-br->radio*2
&& br->pos.y != PANTALLA_ALTO-br->radio*2 ) {
                                    br->direccion = BR_DIR_IZQ_ABAJO;
                              // Limite inferior
                              } else if( br->pos.x != PANTALLA_ANCHO-br->radio*2
&& br->pos.y == PANTALLA_ALTO-br->radio*2 ) {
                                    br->direccion = BR_DIR_DCHA_ARRIBA;
                              }
                        break;
                  }

                  // Asignar la nueva posicion calculada
                  shmBolaRoja[BR_X+desplazamiento] = br->pos.x;
                  shmBolaRoja[BR_Y+desplazamiento] = br->pos.y;
            } else {
                  // Eliminar timer si la bola ya no existe
                  SDL_RemoveTimer(timerIdRoja[br->id]);
                  timerIdRoja[br->id] = (SDL_TimerID)-1;
            }
      }

      return interval;
}

void dibujarBolaRoja() {
      SDL_Rect    rect,
                  srcRect;
      int desplazamiento,
            x,
            y,
            srcX,
            radio;
```

```
	// Este bucle recorre todas las bolas y dibuja cada una de ellas
	for(int i=0; i < NUM_BOLAS_ROJAS; i++) {
		desplazamiento = i*shmTamanoBolaRoja;
		if( shmBolaRoja[BR_ACTIVA + desplazamiento] ) {
			x = shmBolaRoja[BR_X+desplazamiento];
			y = shmBolaRoja[BR_Y+desplazamiento];
			radio = shmBolaRoja[BR_RADIO+desplazamiento];

			if( radio == BR_RADIO_INICIAL ) {
				srcX = BR_POS_1;
			} else if( radio == BR_RADIO_T2 ) {
				srcX = BR_POS_2;
			} else if ( radio == BR_RADIO_T3 ) {
				srcX = BR_POS_3;
			} else if( radio == BR_RADIO_T4 ) {
				srcX = BR_POS_4;
			}

			srcRect = (SDL_Rect) { (Sint16)srcX, (Sint16)0, (Uint16)
(radio*2), (Uint16)(radio*2) };
			rect = (SDL_Rect) { (Sint16)x, (Sint16)y, (Uint16)(radio*2),
(Uint16)(radio*2) };
			SDL_BlitSurface(srfBolaRoja, &srcRect, pantalla, &rect);
		}
	}
}

int crearProcesoBolaAzul() {
	pid_t pid;

	if( pipe(pipePosBA) == -1 ) {
		printf("Error al intentar crear el pipe pipePosBA.\n");
	}

	pid = fork();

	if( pid == 0 ) { //Proceso hijo
		printf("Bola Azul iniciado...\n");
		close(pipePosBA[0]); //Cerrar entrada pipe para hijo

		// Inicializando la semilla de generador de numeros aleatorio
		srand(time(NULL));

		// Colocar la bola azul en un lugar aleatorio en la pantalla
		Point pos;
		pos.x = rand() % (PANTALLA_ANCHO - BA_RADIO*2);
		pos.y = rand() % (PANTALLA_ALTO - BA_RADIO*2);

		// Informamos de su posicion al proceso principal y
		// la activamos.
		write(pipePosBA[1], &pos, sizeof(Point));
		kill(pidPrincipal, SIGMOVAZUL);
		SDL_Delay(20);
		kill(pidPrincipal, SIGTOGGLEAZUL);

		do {
			while(shmJuego[JUEGO_PAUSADO] && !shmJuego[JUEGO_FINAL]) {
				// Si el juego esta pausado este bucle interrumpira la
				// ejecucion.
			}
```

```
                  // Actualizamos la posicion de la bola azul e informamos
                  // al proceso principal.
                  pos.y++;
                  write(pipePosBA[1], &pos, sizeof(Point));
                  SDL_Delay(20);
                  kill(pidPrincipal, SIGMOVAZUL);
            } while(pos.y <= PANTALLA_ALTO && !shmJuego[JUEGO_FINAL]);

            // Mandamos señal al proceso principal para que sepa
            // que la bola azul ya no existe.
            kill(pidPrincipal, SIGTOGGLEAZUL);
            printf("Bola Azul finalizado...\n");
            quick_exit(0);
      } else if( pid == -1 ) {
            perror("Error al crear el proceso Bola Azul.");
            exit(1);
      }

      close(pipePosBA[1]); //Cerrar salida pipe para padre

      return pid;
}

void sigBA(int signal) {
      read(pipePosBA[0], &posAzul, sizeof(Point));
      dibujarBolaAzul();
}

void sigToggleBA(int signal) {
      isVisibleAzul = !isVisibleAzul;
}

void sigLeerBA(int signal) {
      write(pipeIsVisibleBA[1], &isVisibleAzul, 1);
}

void dibujarBolaAzul() {
      SDL_Rect rect;

      rect = (SDL_Rect) { (Sint16)posAzul.x, (Sint16)posAzul.y, (Uint16)
(BA_RADIO*2), (Uint16)(BA_RADIO*2) };
      SDL_BlitSurface(srfBolaAzul, NULL, pantalla, &rect);
}

int crearProcesoBolaNegra() {
      pid_t       pid;

      pid = fork();

      if( pid == 0 ) { //Proceso hijo
            Uint32                  inicio = 0;
            BolaNegra         bn[NUM_BOLAS_NEGRAS];
            int               nextId = 0;

            // Inicializar todas las bolas negras
            for(int i=0; i < NUM_BOLAS_NEGRAS; i++) {
                  timerIdNegra[i] = (SDL_TimerID)-1;
                  inicializarBolaNegra(i, bn);
            }

            // Inicializando timer
```

```
            if (SDL_Init(SDL_INIT_TIMER) == -1) {
                  printf("No se pudo iniciar timer: %s\n", SDL_GetError());
                  exit(1);
            }

            // Introducir un pequeño delay para que el juego no
            // comience inmediatamente y de tiempo a "prepararse".
            SDL_Delay(2000);

            inicio = SDL_GetTicks();

            printf("Bola Negra iniciado...\n");

            while(!shmJuego[JUEGO_FINAL]) {
                  while(shmJuego[JUEGO_PAUSADO] && !shmJuego[JUEGO_FINAL]) {
                        inicio = SDL_GetTicks();
                  }

                  // Cada cierto tiempo inicializar y sacar una nueva bola
negra, si
                  // no se ha excedido el limite de bolas negras en pantalla.
                  if( SDL_GetTicks() - inicio >= (Uint32)(INTERVALO_NEGRAS *
1000) ) {
                        nextId = getIdLibreNegra();
                        if( nextId != -1 ) {
                              inicializarBolaNegra(nextId, bn);
                              shmBolaNegra[BN_ACTIVA +
nextId*shmTamanoBolaNegra] = 1;
                              timerIdNegra[nextId] = SDL_AddTimer(20,
moverBolaNegra, &bn[nextId]);
                        }
                        inicio = SDL_GetTicks();
                  }
            }

            // Eliminar los timers de todas las bolas
            for(int i=0; i < NUM_BOLAS_NEGRAS; i++) {
                  if( timerIdNegra[i] != (SDL_TimerID)-1 )
SDL_RemoveTimer(timerIdNegra[i]);
            }

            printf("Bola Negra finalizado...\n");
            exit(0);
      } else if( pid == -1 ) {
            perror("Error al crear el proceso Bola Negra.");
            exit(1);
      }

      return pid;
}

// Buscar una bola negra libre y retornar su id
int getIdLibreNegra() {
      bool encontrado = false;
      int id = -1;

      for(int i=0; i < NUM_BOLAS_NEGRAS && !encontrado; i++) {
            if( timerIdNegra[i] == (SDL_TimerID)-1 ) {
                  id = i;
                  encontrado = true;
            }
```

```
	}

	return id;
}

void inicializarBolaNegra(int id, BolaNegra *bn) {
	// Inicializando la semilla de generador de numeros aleatorio
	srand(time(NULL));

	bn[id].id = id;
	bn[id].pos.x = rand() % (PANTALLA_ANCHO-BN_RADIO);
	bn[id].pos.y = rand() % (PANTALLA_ALTO-BN_RADIO);
	bn[id].direccion = rand() % 4;
	shmBolaNegra[BN_X + id*shmTamanoBolaNegra] = bn[id].pos.x;
	shmBolaNegra[BN_Y + id*shmTamanoBolaNegra] = bn[id].pos.y;
	shmBolaNegra[BN_ACTIVA + id*shmTamanoBolaNegra] = 0;
}

// Controla el movimiento de las bolas negras
Uint32 moverBolaNegra(Uint32 interval, void *param) {
	if( !shmJuego[JUEGO_PAUSADO] ) {
		BolaNegra *bn = (BolaNegra *)param;
		int desplazamiento = bn->id * shmTamanoBolaNegra;

		if( shmBolaNegra[BN_ACTIVA + desplazamiento] ) {

			switch(bn->direccion) {
				case BN_DIR_IZQ_ARRIBA:
					bn->pos.x = (bn->pos.x > BN_VELOCIDAD) ? bn-
>pos.x-BN_VELOCIDAD : 0;
					bn->pos.y = (bn->pos.y > BN_VELOCIDAD) ? bn-
>pos.y-BN_VELOCIDAD : 0;

					// Si toca lado izquierdo desaparece la bola
negra, sino rebota
					if( bn->pos.x == 0 ) {
						shmBolaNegra[BN_ACTIVA+desplazamiento] = 0;
					} else if( bn->pos.x != 0 && bn->pos.y == 0 ) {
						bn->direccion = BN_DIR_IZQ_ABAJO;
					}
				break;
				case BN_DIR_IZQ_ABAJO:
					bn->pos.x = (bn->pos.x > BN_VELOCIDAD) ? bn-
>pos.x-BN_VELOCIDAD : 0;
					bn->pos.y = (bn->pos.y < PANTALLA_ALTO-
BN_VELOCIDAD-BN_RADIO*2) ? bn->pos.y+BN_VELOCIDAD : PANTALLA_ALTO-BN_RADIO*2;

					// Si toca lado izquierdo desaparece la bola
negra, sino rebota
					if( bn->pos.x == 0 ) {
						shmBolaNegra[BN_ACTIVA+desplazamiento] = 0;
					} else if( bn->pos.x != 0 && bn->pos.y ==
PANTALLA_ALTO-BN_RADIO*2 ) {
						bn->direccion = BN_DIR_IZQ_ARRIBA;
					}
				break;
				case BN_DIR_DCHA_ARRIBA:
					bn->pos.x = (bn->pos.x < PANTALLA_ANCHO-
BN_VELOCIDAD-BN_RADIO*2) ? bn->pos.x+BN_VELOCIDAD : PANTALLA_ANCHO-BN_RADIO*2;
					bn->pos.y = (bn->pos.y > BN_VELOCIDAD) ? bn-
>pos.y-BN_VELOCIDAD : 0;
```

```
                              // Si toca lado derecho desaparece la bola negra,
sino rebota
                              if( bn->pos.x == PANTALLA_ANCHO-BN_RADIO*2 ) {
                                    shmBolaNegra[BN_ACTIVA+desplazamiento] = 0;
                              } else if( bn->pos.x != PANTALLA_ANCHO-BN_RADIO*2
&& bn->pos.y == 0 ) {
                                    bn->direccion = BN_DIR_DCHA_ABAJO;
                              }
                        break;
                        case BN_DIR_DCHA_ABAJO:
                              bn->pos.x = (bn->pos.x < PANTALLA_ANCHO-
BN_VELOCIDAD-BN_RADIO*2) ? bn->pos.x+BN_VELOCIDAD : PANTALLA_ANCHO-BN_RADIO*2;
                              bn->pos.y = (bn->pos.y < PANTALLA_ALTO-
BN_VELOCIDAD-BN_RADIO*2) ? bn->pos.y+BN_VELOCIDAD : PANTALLA_ALTO-BN_RADIO*2;

                              // Si toca lado derecho desaparece la bola negra,
sino rebota
                              if( bn->pos.x == PANTALLA_ANCHO-BN_RADIO*2 ) {
                                    shmBolaNegra[BN_ACTIVA+desplazamiento] = 0;
                              } else if( bn->pos.x != PANTALLA_ANCHO-BN_RADIO*2
&& bn->pos.y == PANTALLA_ALTO-BN_RADIO*2 ) {
                                    bn->direccion = BN_DIR_DCHA_ARRIBA;
                              }
                        break;
                  }

                  // Asignar los valores calculados
                  shmBolaNegra[BN_X+desplazamiento] = bn->pos.x;
                  shmBolaNegra[BN_Y+desplazamiento] = bn->pos.y;
            } else {
                  // Si ya no existe la bola negra, eliminar su timer
                  SDL_RemoveTimer(timerIdNegra[bn->id]);
                  timerIdNegra[bn->id] = (SDL_TimerID)-1;
            }
      }

      return interval;
}

void dibujarBolaNegra() {
      SDL_Rect rect;
      int desplazamiento,
            x,
            y;

      // Dibujar todas las bolas negras que haya en pantalla
      for(int i=0; i < NUM_BOLAS_NEGRAS; i++) {
            desplazamiento = i*shmTamanoBolaNegra;
            if( shmBolaNegra[BN_ACTIVA + desplazamiento] ) {
                  x = shmBolaNegra[BN_X+desplazamiento];
                  y = shmBolaNegra[BN_Y+desplazamiento];

                  rect = (SDL_Rect) { (Sint16)x, (Sint16)y, (Uint16)
(BN_RADIO*2), (Uint16)(BN_RADIO*2) };
                  SDL_BlitSurface(srfBolaNegra, NULL, pantalla, &rect);
            }
      }
}

void inicializar() {
```

```
      if( SDL_Init(SDL_INIT_VIDEO) == -1 ) {
            printf("No se pudo iniciar el video: %s\n", SDL_GetError());
            exit(1);
      }

      if( SDL_Init(SDL_INIT_TIMER) == -1 ) {
            printf("No se pudo iniciar el timer: %s\n", SDL_GetError());
            exit(1);
      }

      if( Mix_OpenAudio( 22050, MIX_DEFAULT_FORMAT, 2, 4096 ) == -1 ) {
        printf("No se pudo iniciar el audio: %s\n", SDL_GetError());
    }

    if( TTF_Init() == -1 ) {
        printf("No se pudo iniciar True Type Font: %s\n", SDL_GetError());
            exit(1);
    }

      pantalla = SDL_SetVideoMode(PANTALLA_ANCHO, PANTALLA_ALTO, 32,
SDL_HWSURFACE | SDL_DOUBLEBUF);
      if(!pantalla) {
            printf("No se pudo iniciar el modo de pantalla: %s\n",
SDL_GetError());
            exit(1);
      }

      fondo = IMG_Load("img/bg.jpg");
      if( !fondo ) {
            printf("No se pudo cargar el fondo.\n");
            exit(1);
      }

      srfFlash = IMG_Load("img/flash.png");
      if( !srfFlash ) {
            printf("No se pudo cargar la imagen del flash.\n");
            exit(1);
      }

      srfEnPausa = IMG_Load("img/pausa.png");
      if( !srfEnPausa ) {
            printf("No se pudo cargar la imagen de pausa.\n");
            exit(1);
      }

      srfHasPerdido = IMG_Load("img/hasperdido.png");
      if( !srfHasPerdido ) {
            printf("No se pudo cargar la imagen que se muestra cuando
pierdes.\n");
            exit(1);
      }

      srfVolumen = IMG_Load("img/volumen.png");
      if( !srfVolumen ) {
            printf("No se pudo cargar la imagen del volumen.\n");
            exit(1);
      }

      srfBolaVerde = IMG_Load("img/bolaverde.png");
      if( !srfBolaVerde ) {
```

```
        printf("No se pudo cargar la imagen de la bola verde.\n");
        exit(1);
  }

  srfBolaRoja = IMG_Load("img/bolaroja.png");
  if( !srfBolaRoja ) {
        printf("No se pudo cargar la imagen de la bola roja.\n");
        exit(1);
  }

  srfBolaPequena = IMG_Load("img/bolapequena.png");
  if( !srfBolaPequena ) {
        printf("No se pudo cargar la imagen de la bola pequeña.\n");
        exit(1);
  }

  srfBolaAzul = IMG_Load("img/bolaazul.png");
  if( !srfBolaAzul ) {
        printf("No se pudo cargar la imagen de la bola azul.\n");
        exit(1);
  }

  srfBolaNegra = IMG_Load("img/bolanegra.png");
  if( !srfBolaNegra ) {
        printf("No se pudo cargar la imagen de la bola negra.\n");
        exit(1);
  }

  colorBlanco = (SDL_Color){ 255, 255, 255 };
  colorRojo = (SDL_Color){ 255, 0, 0 };

  fontPuntuacion = TTF_OpenFont("fonts/american-captain.ttf", 28);
  if( fontPuntuacion == NULL ) {
        printf("No se pudo cargar la fuente American Captain.\n");
        exit(1);
  }

  // MUSICA DEL JUEGO //////////////////////
musica = Mix_LoadMUS("snd/dnbweird.ogg");
if( musica == NULL ) {
  printf("No se pudo cargar la cancion dnbweird.ogg.\n");
}

Mix_VolumeMusic(MIX_MAX_VOLUME / 2);
Mix_Volume(-1, MIX_MAX_VOLUME / 2);

if( Mix_PlayMusic(musica, -1) == -1 ) {
        printf("No se ha podido reproducir la musica.\n");
  }

  shot = Mix_LoadWAV("snd/shot.wav");
  explosion = Mix_LoadWAV("snd/explosion.wav");
  whip = Mix_LoadWAV("snd/whip.wav");


  SDL_WM_SetCaption( "Orb Attack por Francisco Mendez Vilas", NULL );

  SDL_EnableKeyRepeat(200, 0);

  // LOGICA DEL JUEGO //////////////////////
```

```
	shmJuego[JUEGO_FINAL] = 0;
	shmJuego[JUEGO_PAUSADO] = 0;
	shmJuego[JUEGO_PUNTOS] = ENERGIA_BOLA_VERDE;
}

// Carga la configuracion del juego desde el archivo config.txt
void cargarConfiguracion() {
	ifstream fichero("config.txt");
	string linea;
	char  *clave;
	int   valor,
		contador = 0;

	// Lee linea a linea e interpreta el valor que se
	// debe asignar y a que.
	while(contador < 13 && getline(fichero, linea)) {
	    clave = strtok((char *)linea.c_str(), " \t");
	      valor = atoi(strtok(NULL, " \t"));

	      if( strcmp(clave, "energia_bolas_verdes") == 0 ) {
	            ENERGIA_BOLA_VERDE = valor;
	      } else if( strcmp(clave, "num_bolas_rojas") == 0 ) {
	            NUM_BOLAS_ROJAS = valor;
	      } else if( strcmp(clave, "intervalo_rojas") == 0 ) {
	            INTERVALO_ROJAS = valor;
	      } else if( strcmp(clave, "intervalo_azules") == 0 ) {
	            INTERVALO_AZUL = valor;
	      } else if( strcmp(clave, "num_bolas_negras") == 0 ) {
	            NUM_BOLAS_NEGRAS = valor;
	      } else if( strcmp(clave, "intervalo_negras") == 0 ) {
	            INTERVALO_NEGRAS = valor;
	      } else if( strcmp(clave, "bolas_verdes_pequeñas") == 0 ) {
	            NUM_BOLAS_PEQUENAS = valor;
	      } else if( strcmp(clave, "puntuacion_t1_roja") == 0 ) {
	            PUNTUACION_T1_ROJA = valor;
	      } else if( strcmp(clave, "puntuacion_t2_roja") == 0 ) {
	            PUNTUACION_T2_ROJA = valor;
	      } else if( strcmp(clave, "puntuacion_t3_roja") == 0 ) {
	            PUNTUACION_T3_ROJA = valor;
	      } else if( strcmp(clave, "puntuacion_t4_roja") == 0 ) {
	            PUNTUACION_T4_ROJA = valor;
	      } else if( strcmp(clave, "puntuacion_bola_negra") == 0 ) {
	            PUNTUACION_NEGRA = valor;
	      } else if( strcmp(clave, "quita_energia_roja") == 0 ) {
	            QUITA_ENERGIA_ROJA = valor;
	      }

	      contador++;
	}

	// Reservamos memoria para NUM_BOLAS_X dado que en la declaracion
	// del array en el fichero .h no podiamos saber aun este numero porque
	// aun tenia que ser leido del fichero.
	timerIdRoja = (SDL_TimerID *)malloc(sizeof(SDL_TimerID) *
NUM_BOLAS_ROJAS);
	timerIdNegra = (SDL_TimerID *)malloc(sizeof(SDL_TimerID) *
NUM_BOLAS_NEGRAS);
}
```

```
// Menu inicial
void mostrarPantallaInicio() {
      bool continuar = false;
      SDL_Rect rect;
      SDL_Event event;
      SDL_Surface *imagen;

      if( SDL_Init(SDL_INIT_VIDEO) == -1 ) {
            printf("No se pudo iniciar el video: %s\n", SDL_GetError());
            exit(1);
      }

      if( Mix_OpenAudio( 22050, MIX_DEFAULT_FORMAT, 2, 4096 ) == -1 ) {
        printf("No se pudo iniciar el mezclador de Audio: %s\n",
SDL_GetError());
    }

    musicaInicio = Mix_LoadMUS("snd/funkyphresh.ogg");
    if( musicaInicio == NULL ) {
      printf("No se pudo cargar la cancion funkyphresh.ogg.\n");
    }

      SDL_WM_SetCaption( "Orb Attack por Francisco Mendez Vilas", NULL );

      pantalla = SDL_SetVideoMode(PANTALLA_ANCHO, PANTALLA_ALTO, 32,
SDL_HWSURFACE | SDL_DOUBLEBUF);
      if(!pantalla) {
            printf("No se pudo iniciar el modo de pantalla: %s\n",
SDL_GetError());
            exit(1);
      }

      srfInicio = IMG_Load("img/inicio.jpg");
      if( !srfInicio ) {
            printf("No se pudo cargar la imagen de inicio.\n");
            exit(1);
      }

      srfInstrucciones = IMG_Load("img/instrucciones.jpg");
      if( !srfInstrucciones ) {
            printf("No se pudo cargar la imagen de instrucciones.\n");
            exit(1);
      }

      imagen = srfInicio;

      Mix_VolumeMusic(MIX_MAX_VOLUME / 2);
      Mix_FadeInMusic(musicaInicio, -1, 2000);

      do {
            rect = (SDL_Rect) { 0, 0, PANTALLA_ANCHO, PANTALLA_ALTO };
            SDL_BlitSurface(imagen , NULL, pantalla, &rect);
            SDL_Flip(pantalla);

            SDL_WaitEvent(&event);
            if( event.type == SDL_KEYDOWN ) {
                  switch(event.key.keysym.sym) {
                        case SDLK_ESCAPE:
                              exit(0);
                        break;
                        case SDLK_RETURN:
```

```
                        continuar = true;
                    break;
                    case SDLK_i:
                        if( imagen == srfInicio ) {
                            imagen = srfInstrucciones;
                        } else {
                            imagen = srfInicio;
                        }
                    break;
                    default:
                        // No hacer nada (dejar para prevenir Warnings)
                    break;
                }
            } else if( event.type == SDL_QUIT ) {
                exit(0);
            }
        } while(!continuar);

        Mix_FadeOutMusic(1000);
        SDL_Delay(1000);

        SDL_FreeSurface(srfInicio);
        SDL_FreeSurface(srfInstrucciones);
        Mix_FreeMusic(musicaInicio);
        Mix_CloseAudio();
        SDL_Quit();
}

void dibujarFondo() {
        SDL_Rect rect;

        rect = (SDL_Rect) { 0, 0, PANTALLA_ANCHO, PANTALLA_ALTO };
        SDL_BlitSurface(fondo , NULL, pantalla, &rect);
}

void dibujarFlash() {
        SDL_Rect rect;

        rect = (SDL_Rect) { 0, 0, PANTALLA_ANCHO, PANTALLA_ALTO };
        SDL_BlitSurface(srfFlash , NULL, pantalla, &rect);
}

void dibujarEnPausa() {
        SDL_Rect rect;

        rect = (SDL_Rect) { 0, 0, PANTALLA_ANCHO, PANTALLA_ALTO };
        SDL_BlitSurface(srfEnPausa , NULL, pantalla, &rect);
}

void dibujarHasPerdido() {
        SDL_Rect rect;

        rect = (SDL_Rect) { 0, 0, PANTALLA_ANCHO, PANTALLA_ALTO };
        SDL_BlitSurface(srfHasPerdido , NULL, pantalla, &rect);
}

void dibujarPuntuacion() {
        SDL_Rect rect;
        char strPuntuacion[20];
        SDL_Color colorTexto;
```

```
	colorTexto = shmJuego[JUEGO_PUNTOS] < 200 ? colorRojo : colorBlanco;

	sprintf(strPuntuacion, "Puntos: %d", shmJuego[JUEGO_PUNTOS]);
	srfPuntuacion = TTF_RenderText_Blended(fontPuntuacion, strPuntuacion,
colorTexto);

	rect = (SDL_Rect) { (Sint16)10, (Sint16)10, PANTALLA_ANCHO-20, 30 };
	SDL_BlitSurface(srfPuntuacion, NULL, pantalla, &rect);
}

void dibujarVolumen() {
	SDL_Rect	rect,
				rect2;
	Sint16 altura;
	int volumen = Mix_VolumeMusic(-1);

	if( volumen == MIX_MAX_VOLUME ) {
		altura = 0;
	} else if( volumen >= (MIX_MAX_VOLUME / 4) * 3 ) {
		altura = 27;
	} else if( volumen >= MIX_MAX_VOLUME / 2 ) {
		altura = 54;
	} else if( volumen > 0 ) {
		altura = 81;
	} else {
		altura = 108;
	}

	rect = (SDL_Rect) { (Sint16)(PANTALLA_ANCHO-140), (Sint16)10, (Uint32)130,
(Uint32)25 };
	rect2 = (SDL_Rect) { (Sint16)0, altura, (Uint32)130, (Uint32)27 };
	SDL_BlitSurface(srfVolumen , &rect2, pantalla, &rect);
}


// Determinar cual es la bola roja que esta mas cerca de la verde
int bolaRojaMasCercana() {
	double	distancia = -1,
			aux;
	int	id = -1;

	// Una por una determina la distancia entre la bola roja y la
	// bola verde, quedandose con la distancia mas corta.
	for(int i=0; i < NUM_BOLAS_ROJAS; i++) {
		if( shmBolaRoja[BR_ACTIVA + i * shmTamanoBolaRoja] ) {
			aux = distanciaBolaVerdeBolaRoja(i);

			if( distancia != -1 && aux < distancia ) {
				id = i;
			} else if( distancia == -1 ) {
				distancia = aux;
				id = i;
			}
		}
	}

	return id;
}

// Detectar si hay colision entre la bola verde y alguna
// bola roja.
```

```
bool detectarColisionBVBR() {
	double	distancia;
	int	radioRoja;
	bool	hayColision = false;

	for(int i=0; i < NUM_BOLAS_ROJAS; i++) {
		if( shmBolaRoja[BR_ACTIVA + i * shmTamanoBolaRoja] ) {
			radioRoja = shmBolaRoja[BR_RADIO + i * shmTamanoBolaRoja];
			distancia = distanciaBolaVerdeBolaRoja(i);

			if( distancia <= radioRoja + BV_RADIO ) {
				shmJuego[JUEGO_PUNTOS] -= QUITA_ENERGIA_ROJA;
				hayColision = true;
			}
		}
	}

	return hayColision;
}

// Calcular distancia entre la bola verde y la bola roja con id idRoja
double distanciaBolaVerdeBolaRoja(int idRoja) {
	Point p1,
		p2;
	int	radioRoja,
		desplazamientoRoja;
	double	distancia;

	desplazamientoRoja = shmTamanoBolaRoja * idRoja;
	radioRoja = shmBolaRoja[BR_RADIO + desplazamientoRoja];
	p1.x = shmBolaRoja[BR_X + desplazamientoRoja] + radioRoja;
	p1.y = shmBolaRoja[BR_Y + desplazamientoRoja] + radioRoja;
	p2.x = shmBolaVerde[BV_X] + BV_RADIO;
	p2.y = shmBolaVerde[BV_Y] + BV_RADIO;
	distancia = sqrt( pow(abs(p2.x - p1.x),2) + pow(abs(p2.y - p1.y),2) );

	return distancia;
}


// Detectar colision entre alguna bola pequeña y alguna bola roja
void detectarColisionBPBR() {
	double	distancia;
	int	radioRoja;

	for(int iBP=0; iBP < NUM_BOLAS_PEQUENAS; iBP++) {
		if( bolasPequenas[iBP].pid ) {
			for(int i=0; i < NUM_BOLAS_ROJAS; i++) {
				if( shmBolaRoja[BR_ACTIVA + i * shmTamanoBolaRoja] ) {
					radioRoja = shmBolaRoja[BR_RADIO + i *
shmTamanoBolaRoja];
					distancia = distanciaBolaPequenaBolaRoja(iBP, i);

					if( distancia <= radioRoja + BP_RADIO ) {
						kill(bolasPequenas[iBP].pid, SIGKILL);
						bolasPequenas[iBP].pid = 0;
						bpTotal--;
						shmBolaRoja[BR_ACTIVA + i *
shmTamanoBolaRoja] = 0;
						Mix_PlayChannel(-1, explosion, 0);
```

```
                                    // Dependiendo del tamaño de la bola roja
                                    // valdra mas puntos o menos.
                                    switch(radioRoja) {
                                          case BR_RADIO_INICIAL:
                                                shmJuego[JUEGO_PUNTOS] +=
PUNTUACION_T1_ROJA;
                                          break;
                                          case BR_RADIO_T2:
                                                shmJuego[JUEGO_PUNTOS] +=
PUNTUACION_T2_ROJA;
                                          break;
                                          case BR_RADIO_T3:
                                                shmJuego[JUEGO_PUNTOS] +=
PUNTUACION_T3_ROJA;
                                          break;
                                          case BR_RADIO_T4:
                                                shmJuego[JUEGO_PUNTOS] +=
PUNTUACION_T4_ROJA;
                                          break;
                                    }
                              }
                        }
                  }
            }
      }
}


// Calcular la distancia entre la bola pequeña con id idBP y la bola
// roja con id idRoja.
double distanciaBolaPequenaBolaRoja(int idBP, int idRoja) {
      Point p1,
                  p2;
      int   radioRoja,
                  desplazamiento;
      double      distancia;

      desplazamiento = shmTamanoBolaRoja * idRoja;
      radioRoja = shmBolaRoja[BR_RADIO + desplazamiento];
      p1.x = shmBolaRoja[BR_X + desplazamiento] + radioRoja;
      p1.y = shmBolaRoja[BR_Y + desplazamiento] + radioRoja;
      p2.x = shmBolaPequena[BP_X + idBP * shmTamanoBolaPequena] + BP_RADIO;
      p2.y = shmBolaPequena[BP_Y + idBP * shmTamanoBolaPequena] + BP_RADIO;
      distancia = sqrt( pow(abs(p2.x - p1.x),2) + pow(abs(p2.y - p1.y),2) );

      return distancia;
}


// Detectar colision entre alguna bola pqueña y la bola azul
void detectarColisionBPBA() {
      if( isVisibleAzul ) {
            double distancia;
            Point p1,
                        p2;

            for(int i=0; i < NUM_BOLAS_PEQUENAS; i++) {
                  if( bolasPequenas[i].pid ) {
                        p1.x = posAzul.x + BA_RADIO;
                        p1.y = posAzul.y + BA_RADIO;
                        p2.x = shmBolaPequena[BP_X + i*shmTamanoBolaPequena] +
```

```
BP_RADIO;
                        p2.y = shmBolaPequena[BP_Y + i*shmTamanoBolaPequena] +
BP_RADIO;

                        distancia = sqrt( pow(abs(p2.x - p1.x),2) + pow(abs(p2.y
- p1.y),2) );

                        if( distancia <= BA_RADIO + BP_RADIO ) {
                              kill(bolasPequenas[i].pid, SIGKILL);
                              bolasPequenas[i].pid = 0;
                              bpTotal--;
                              if( pidBolaAzul ) kill(pidBolaAzul, SIGKILL);
                              pidBolaAzul = 0;
                              isVisibleAzul = false;
                              Mix_PlayChannel(-1, whip, 0);
                        }
                  }
            }
      }
}

// Detectar si hay colision entre alguna bola negra y alguna bola roja
int detectarColisionBNBR() {
      double      distancia;
      int   radioRoja;

      for(int i=0; i < NUM_BOLAS_NEGRAS; i++) {
            for(int h=0; h < NUM_BOLAS_ROJAS; h++) {
                  if( shmBolaRoja[BR_ACTIVA + h * shmTamanoBolaRoja]
                        && shmBolaNegra[BN_ACTIVA + i * shmTamanoBolaNegra] )
                  {
                        radioRoja = shmBolaRoja[BR_RADIO + h *
shmTamanoBolaRoja];
                        distancia = distanciaBolaNegraBolaRoja(i, h);

                        if( distancia <= radioRoja + BN_RADIO ) {
                              shmBolaNegra[BN_ACTIVA + i * shmTamanoBolaNegra] =
0;
                              shmJuego[JUEGO_PUNTOS] += PUNTUACION_NEGRA;

                              Mix_PlayChannel(-1, explosion, 0);
                              return h;
                        }
                  }
            }

      }

      return -1;
}

// Calcular distancia entre la bola negra con id idNegra y la
// bola roja con id idRoja.
double distanciaBolaNegraBolaRoja(int idNegra, int idRoja) {
      Point p1,
                  p2;
      int   radioRoja,
                  desplazamientoRoja,
                  desplazamientoNegra;
      double      distancia;
```

```
      desplazamientoRoja = shmTamanoBolaRoja * idRoja;
      radioRoja = shmBolaRoja[BR_RADIO + desplazamientoRoja];
      p1.x = shmBolaRoja[BR_X + desplazamientoRoja] + radioRoja;
      p1.y = shmBolaRoja[BR_Y + desplazamientoRoja] + radioRoja;
      desplazamientoNegra = shmTamanoBolaNegra * idNegra;
      p2.x = shmBolaNegra[BN_X + desplazamientoNegra] + BN_RADIO;
      p2.y = shmBolaNegra[BN_Y + desplazamientoNegra] + BN_RADIO;
      distancia = sqrt( pow(abs(p2.x - p1.x),2) + pow(abs(p2.y - p1.y),2) );

      return distancia;
}

// Liberar los recursos SDL y de memoria compartida.
void liberar() {
      printf("\nTerminando (liberando recursos)...\n");
      shmdt((char *)shmJuego);
      shmctl(shmIdJuego, IPC_RMID, 0);
      shmdt((char *)shmBolaVerde);
      shmctl(shmIdBolaVerde, IPC_RMID, 0);
      shmdt((char *)shmBolaRoja);
      shmctl(shmIdBolaRoja, IPC_RMID, 0);
      shmdt((char *)shmBolaPequena);
      shmctl(shmIdBolaPequena, IPC_RMID, 0);
      shmdt((char *)shmBolaNegra);
      shmctl(shmIdBolaNegra, IPC_RMID, 0);
      SDL_FreeSurface(srfFlash);
      SDL_FreeSurface(fondo);
      SDL_FreeSurface(srfBolaVerde);
      SDL_FreeSurface(srfBolaRoja);
      SDL_FreeSurface(srfBolaPequena);
      SDL_FreeSurface(srfBolaAzul);
      SDL_FreeSurface(srfBolaNegra);
      SDL_FreeSurface(srfPuntuacion);
      SDL_FreeSurface(srfVolumen);
      SDL_FreeSurface(srfEnPausa);
      SDL_FreeSurface(srfHasPerdido);
    TTF_CloseFont(fontPuntuacion);
    Mix_FreeMusic(musica);
    Mix_FreeChunk(shot);
    Mix_FreeChunk(explosion);
    Mix_FreeChunk(whip);
    Mix_CloseAudio();
      SDL_Quit();
}
```

Código sin comentarios

P2.h

```
using namespace std;

struct Point {
	int	x;
	int	y;
};

struct PaqueteBP {
	char  orden[100];
	pid_t	pid = 0;
	Point pos = (Point){-1,-1};
	char  direccion = 0;
};

struct BolaPequena {
	pid_t		pid;
	Point		pos;
	char		direccion;
};

struct BolaRoja {
	int			id;
	Point		pos;
	int			radio;
	char		direccion;
};

struct BolaNegra {
	int			id;
	Point		pos;
	char		direccion;
};

#define PI				3.14159265
#define SIGTECLA		54
#define SIGMOVAZUL			SIGUSR1
#define SIGTOGGLEAZUL		SIGUSR2
#define SIGLEERAZUL			52
#define SIGCOMPEQUENA		51
#define SIGDESTRUIRPEQUENA	50
#define SIGPRINCIPALPEQUENA	49

#define	FRAMES_PER_SECOND 200
#define	PANTALLA_ANCHO		800
#define	PANTALLA_ALTO		600

#define KEYS_PER_SECOND		50

#define JUEGO_FINAL			0
#define JUEGO_PAUSADO		1
#define JUEGO_PUNTOS		2


#define	BV_X			0
#define	BV_Y			sizeof(int)
```

```
#define BV_RADIO              30
#define PUNTOS_PERMANENCIA    10

#define BP_X                        0
#define BP_Y                        sizeof(int)
#define BP_RADIO              14

#define TIEMPO_CRECIMIENTO    3
#define     BR_X                    0
#define     BR_Y                    sizeof(int)
#define     BR_RADIO                sizeof(int) * 2
#define BR_ACTIVA             sizeof(int) * 3
#define     BR_RADIO_INICIAL  30
#define     BR_RADIO_T2             40
#define     BR_RADIO_T3             50
#define     BR_RADIO_T4             60
#define     BR_DIR_IZQ_ARRIBA 0
#define     BR_DIR_IZQ_ABAJO  1
#define     BR_DIR_DCHA_ARRIBA      2
#define     BR_DIR_DCHA_ABAJO 3
#define BR_VELOCIDAD          3
#define BR_POS_1              0
#define BR_POS_2              80
#define BR_POS_3              170
#define BR_POS_4              280

#define BA_X                        0
#define BA_Y                        sizeof(int)
#define BA_ACTIVA             sizeof(int) * 2
#define BA_RADIO              30

#define BN_X                        0
#define BN_Y                        sizeof(int)
#define BN_ACTIVA             sizeof(int) * 2
#define BN_RADIO              30
#define     BN_DIR_IZQ_ARRIBA 0
#define     BN_DIR_IZQ_ABAJO  1
#define     BN_DIR_DCHA_ARRIBA      2
#define     BN_DIR_DCHA_ABAJO 3
#define BN_VELOCIDAD          1

int         ENERGIA_BOLA_VERDE,
            NUM_BOLAS_ROJAS,
            INTERVALO_ROJAS,
            INTERVALO_AZUL,
            NUM_BOLAS_NEGRAS,
            INTERVALO_NEGRAS,
            NUM_BOLAS_PEQUENAS,
            PUNTUACION_T1_ROJA,
            PUNTUACION_T2_ROJA,
            PUNTUACION_T3_ROJA,
            PUNTUACION_T4_ROJA,
            PUNTUACION_NEGRA,
            QUITA_ENERGIA_ROJA;

int   pidPrincipal,
            pidBolaVerde,
            pidBolaRoja,
            pidBolaPequena,
            pidBolaAzul,
            pidBolaNegra;
```

```
long int    shmIdJuego;
int               *shmJuego;

long int    shmIdBolaVerde;
int               *shmBolaVerde;
char        tecla;
Point       posBV;
int         pipeTeclaBV[2];

long int    shmIdBolaRoja;
int               *shmBolaRoja;
int         shmTamanoBolaRoja;
SDL_TimerID *timerIdRoja;

BolaPequena bolasPequenas[5];
BolaPequena bp;
int         bpUltimoId;
int         bpTotal;
long int    shmIdBolaPequena;
int               *shmBolaPequena;
int         shmTamanoBolaPequena;

int         pipePosBA[2];
int         pipeIsVisibleBA[2];
Point       posAzul;
bool        isVisibleAzul;

long int    shmIdBolaNegra;
int               *shmBolaNegra;
int         shmTamanoBolaNegra;
SDL_TimerID *timerIdNegra;

SDL_Surface       *pantalla,
                        *srfInicio,
                        *srfInstrucciones,
                        *fondo,
                        *srfFlash,
                        *srfEnPausa,
                        *srfHasPerdido,
                        *srfBolaVerde,
                        *srfBolaRoja,
                        *srfBolaPequena,
                        *srfBolaAzul,
                        *srfBolaNegra,
                        *srfPuntuacion,
                        *srfVolumen;

TTF_Font          *fontPuntuacion;
SDL_Color         colorBlanco,
                        colorRojo;

Mix_Music         *musica,
                        *musicaInicio;

Mix_Chunk         *shot,
                        *explosion,
                        *whip;

int main();
void procesoPrincipal();
```

```
void registerSigMovAzul();
void registerSigToggleAzul();
void registerSigLeerAzul();
void registerSigComPequena();
void registerSigDestruirPequena();
int crearProcesoBolaVerde();
void gestionarTeclaBolaVerde(int);
void dibujarBolaVerde(int, int);
int crearProcesoBolaPequena(int, char);
void sigComPequena(int);
void sigDestruirPequena(int, siginfo_t*, void*);
int obtenerNuevoIdPequena();
void registerSigPrincipalPequena();
void dibujarBolaPequena();
int crearProcesoBolaRoja();
int getIdLibreRoja();
void inicializarBolaRoja(int, BolaRoja*);
Uint32 moverBolaRoja(Uint32, void*);
void dibujarBolaRoja();
int crearProcesoBolaAzul();
void sigBA(int);
void sigToggleBA(int);
void sigLeerBA(int);
void dibujarBolaAzul();
int crearProcesoBolaNegra();
int getIdLibreNegra();
void inicializarBolaNegra(int, BolaNegra*);
Uint32 moverBolaNegra(Uint32, void*);
void dibujarBolaNegra();
void inicializar();
void cargarConfiguracion();
void mostrarPantallaInicio();
void dibujarFondo();
void dibujarFlash();
void dibujarEnPausa();
void dibujarHasPerdido();
void dibujarPuntuacion();
void dibujarVolumen();
int bolaRojaMasCercana();
bool detectarColisionBVBR();
double distanciaBolaVerdeBolaRoja(int);
void detectarColisionBPBR();
double distanciaBolaPequenaBolaRoja(int, int);
void detectarColisionBPBA();
int detectarColisionBNBR();
double distanciaBolaNegraBolaRoja(int, int);
void liberar();
int* crearVariosProcesos();
```

P2.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <signal.h>
#include <string.h>
#include <errno.h>
#include <time.h>
#include <cmath>
```

```
#include <fstream>
#include <sstream>
#include <sys/types.h>
#include <sys/wait.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/time.h>
#include <SDL.h>
#include <SDL_image.h>
#include <SDL_mixer.h>
#include <SDL_ttf.h>

#include "p2.h"

int main() {
	mostrarPantallaInicio();
	cargarConfiguracion();

	shmIdJuego = shmget( (key_t)ftok("/bin/ls", 11), sizeof(long int) + 2,
0777 | IPC_CREAT );
	shmJuego = (int *)shmat(shmIdJuego, NULL, 0);
	memset(shmJuego, 0, sizeof(long int) + 2);

	shmIdBolaVerde = shmget( (key_t)ftok("/bin/ls", 22), sizeof(int)*2, 0777 |
IPC_CREAT );
	shmBolaVerde = (int *)shmat(shmIdBolaVerde, NULL, 0);
	memset(shmBolaVerde, 0, sizeof(int)*2);

	shmTamanoBolaRoja = sizeof(int) * 3 + 1;
	shmIdBolaRoja = shmget( (key_t)ftok("/bin/ls", 33), shmTamanoBolaRoja *
NUM_BOLAS_ROJAS, 0777 | IPC_CREAT );
	shmBolaRoja = (int *)shmat(shmIdBolaRoja, NULL, 0);
	memset(shmBolaRoja, 0, shmTamanoBolaRoja);

	shmTamanoBolaPequena = sizeof(int) * 2 * NUM_BOLAS_PEQUENAS;
	shmIdBolaPequena = shmget( (key_t)ftok("/bin/ls", 33),
shmTamanoBolaPequena, 0777 | IPC_CREAT );
	shmBolaPequena = (int *)shmat(shmIdBolaPequena, NULL, 0);
	memset(shmBolaPequena, 0, shmTamanoBolaPequena);

	shmTamanoBolaNegra = sizeof(int) * 2 + 1;
	shmIdBolaNegra = shmget( (key_t)ftok("/bin/ls", 66), shmTamanoBolaNegra *
NUM_BOLAS_NEGRAS, 0777 | IPC_CREAT );
	shmBolaNegra = (int *)shmat(shmIdBolaNegra, NULL, 0);
	memset(shmBolaNegra, 0, shmTamanoBolaNegra);


	registerSigMovAzul();
	registerSigToggleAzul();
	registerSigLeerAzul();


	registerSigComPequena();
	registerSigDestruirPequena();

	pidPrincipal = getpid();

	pidBolaVerde = crearProcesoBolaVerde();

	if( pidBolaVerde > 0 ) {
		pidBolaRoja = crearProcesoBolaRoja();
```

```
            if( pidBolaRoja > 0 ) {
                  pidBolaNegra = crearProcesoBolaNegra();

                  if( pidBolaNegra > 0 ) {
                        procesoPrincipal();
                  }
            }
      }


      kill(pidBolaNegra, SIGKILL);
      kill(pidBolaAzul, SIGKILL);
      kill(pidBolaRoja, SIGKILL);
      kill(pidBolaVerde, SIGKILL);

      return 0;
}

void procesoPrincipal() {
      int               frame = 0,
                              idRoja;
      SDL_Event         event;
      Uint32                  fpsStart,
                              contadorAzul = SDL_GetTicks(),
                              retardoTeclaMusica = SDL_GetTicks(),
                              retardoTeclaPausa = SDL_GetTicks(),
                              retardoTeclaEnter = SDL_GetTicks();


      inicializar();



      isVisibleAzul = false;
      for(int i=0; i < NUM_BOLAS_PEQUENAS; i++) {
            bolasPequenas[i].pid = 0;
      }
      bpUltimoId = 0;
      bpTotal = 0;

      do {


            SDL_PollEvent(&event);

            if( event.type == SDL_KEYDOWN ) {
                  tecla = event.key.keysym.sym;

                  if( event.key.keysym.sym == SDLK_p ) {
                        if( SDL_GetTicks() - retardoTeclaPausa >= 800 ) {
                              if( shmJuego[JUEGO_PAUSADO] == 0 ) {
                                    shmJuego[JUEGO_PAUSADO] = 1;
                                    Mix_PauseMusic();
                                    dibujarEnPausa();
                                    SDL_Flip(pantalla);
                              } else {
                                    shmJuego[JUEGO_PAUSADO] = 0;
                                    Mix_ResumeMusic();
                              }
```

```
                        retardoTeclaPausa = SDL_GetTicks();
                  }
            } else if( event.key.keysym.sym == SDLK_m ) {
                  if( Mix_PlayingMusic() && SDL_GetTicks() -
retardoTeclaMusica >= 800 ) {
                        if( Mix_PausedMusic() ) {
                              Mix_ResumeMusic();
                        } else {
                              Mix_PauseMusic();
                        }

                        retardoTeclaMusica = SDL_GetTicks();
                  }

            } else if( event.key.keysym.sym == SDLK_RETURN ) {
                  if( SDL_GetTicks() - retardoTeclaEnter >= 300 ) {
                        kill(pidBolaVerde, SIGTECLA);
                        SDL_Delay(1);
                        write(pipeTeclaBV[1], &tecla, sizeof(char));
                        retardoTeclaEnter = SDL_GetTicks();
                  }
            }


            if( frame % KEYS_PER_SECOND == 0 ) {
                  switch(event.key.keysym.sym) {
                        case SDLK_PLUS:
                              if( Mix_VolumeMusic(-1) != MIX_MAX_VOLUME )
{
                                    Mix_VolumeMusic(Mix_VolumeMusic(-1) +
1);
                                    Mix_Volume(-1, Mix_VolumeMusic(-1) +
1);
                              }
                        break;
                        case SDLK_MINUS:
                              if( Mix_VolumeMusic(-1) != 0 ) {
                                    Mix_VolumeMusic(Mix_VolumeMusic(-1) -
1);
                                    Mix_Volume(-1, Mix_VolumeMusic(-1) -
1);
                              }
                        break;
                        case SDLK_RETURN:


                        break;
                        default:
                              kill(pidBolaVerde, SIGTECLA);
                              SDL_Delay(1);
                              write(pipeTeclaBV[1], &tecla,
sizeof(char));
                        break;
                  }
            }

      } else if( event.type == SDL_QUIT ) {
            shmJuego[JUEGO_FINAL] = 1;
      }
```

```
            if( !shmJuego[JUEGO_PAUSADO] ) {


                  if( frame % FRAMES_PER_SECOND == 0 ) {
                        dibujarFondo();


                        detectarColisionBPBR();


                        idRoja = detectarColisionBNBR();
                        if( idRoja != -1 ) {
                              shmBolaRoja[BR_ACTIVA + idRoja *
shmTamanoBolaRoja] = 0;
                        }


                        detectarColisionBPBA();


                        if( detectarColisionBVBR() ) {
                              dibujarFlash();
                        }


                        if( isVisibleAzul ) dibujarBolaAzul();
                        dibujarBolaVerde(shmBolaVerde[BV_X],
shmBolaVerde[BV_Y]);
                        dibujarBolaRoja();
                        dibujarBolaNegra();
                        dibujarBolaPequena();



                        if( shmJuego[JUEGO_PUNTOS] <= 0 ) {
                              shmJuego[JUEGO_PUNTOS] = 0;
                              shmJuego[JUEGO_FINAL] = 1;
                        }


                        dibujarPuntuacion();
                        dibujarVolumen();


                        SDL_Flip(pantalla);
                        frame = 0;
                  }


                  if( SDL_GetTicks() - fpsStart < 1000 ) {
                  frame++;
               } else {
                  fpsStart = SDL_GetTicks();
               }



            if( !isVisibleAzul ) {
                  if( SDL_GetTicks() - contadorAzul >=
```

```
(Uint32)INTERVALO_AZUL*1000 ) {
                  contadorAzul = SDL_GetTicks();
                  pidBolaAzul = crearProcesoBolaAzul();
                  }
            } else {
                  contadorAzul = SDL_GetTicks();
            }

          }


      } while(event.key.keysym.sym != SDLK_ESCAPE && shmJuego[JUEGO_FINAL] ==
0);



      shmJuego[JUEGO_FINAL] = 1;

      if( shmJuego[JUEGO_PUNTOS] <= 0 ) {
            Mix_HaltMusic();
            dibujarHasPerdido();
            SDL_Flip(pantalla);
            SDL_Delay(4000);
            SDL_WaitEvent(NULL);
      } else {
            Mix_FadeOutMusic(1000);
            SDL_Delay(1000);
      }
}

void registerSigMovAzul() {

      struct sigaction act;
      act.sa_handler = &sigBA;
      act.sa_flags = SA_RESTART;
      sigemptyset(&act.sa_mask);
      sigaction(SIGMOVAZUL, &act, NULL);
}

void registerSigToggleAzul() {

      struct sigaction act;
      act.sa_handler = &sigToggleBA;
      act.sa_flags = SA_RESTART;
      sigemptyset(&act.sa_mask);
      sigaction(SIGTOGGLEAZUL, &act, NULL);
}

void registerSigLeerAzul() {

      struct sigaction act;
      act.sa_handler = &sigLeerBA;
      act.sa_flags = SA_RESTART;
      sigemptyset(&act.sa_mask);
      sigaction(SIGLEERAZUL, &act, NULL);
}

void registerSigComPequena() {

      struct sigaction act;
      act.sa_handler = &sigComPequena;
```

```
	act.sa_flags = SA_RESTART;
	sigemptyset(&act.sa_mask);
	sigaction(SIGCOMPEQUENA, &act, NULL);
}

void registerSigDestruirPequena() {

	struct sigaction act;
	act.sa_sigaction = &sigDestruirPequena;
	act.sa_flags = SA_RESTART | SA_SIGINFO;
	sigemptyset(&act.sa_mask);
	sigaction(SIGDESTRUIRPEQUENA, &act, NULL);
}

int crearProcesoBolaVerde() {
	pid_t pid;



	if( pipe(pipeTeclaBV) == -1 ) {
		printf("Error al crear pipe pipeTeclaBV.\n");
	}



	if( pipe(pipeIsVisibleBA) == -1 ) {
		printf("Error al crear pipe pipeIsVisibleBA.\n");
	}

	pid = fork();

	if( pid == 0 ) {
		close(pipeTeclaBV[1]);
		close(pipeIsVisibleBA[1]);

		struct sigaction act;
		act.sa_handler = &gestionarTeclaBolaVerde;
		act.sa_flags = SA_RESTART;
		sigemptyset(&act.sa_mask);
		sigaction(SIGTECLA, &act, NULL);

		Uint32 inicio = SDL_GetTicks();



		posBV.x = PANTALLA_ANCHO / 2;
		posBV.y = PANTALLA_ALTO / 2;
		shmBolaVerde[BV_X] = posBV.x;
		shmBolaVerde[BV_Y] = posBV.y;

		printf("Bola Verde iniciado...\n");
		do {
			if( !shmJuego[JUEGO_PAUSADO] ) {
				if( SDL_GetTicks() - inicio >= 1000 ) {
					shmJuego[JUEGO_PUNTOS] += PUNTOS_PERMANENCIA;
					inicio = SDL_GetTicks();
				}
			} else {
				inicio = SDL_GetTicks();
			}
		} while(!shmJuego[JUEGO_FINAL]);
```

```
            printf("Bola Verde finalizado...\n");
            exit(0);
      } else if( pid == -1 ) {
            perror("Error al crear el proceso Bola Verde.");
            exit(1);
      }

      close(pipeTeclaBV[0]);
      close(pipeIsVisibleBA[0]);
      return pid;
}



void gestionarTeclaBolaVerde(int signal) {
      read(pipeTeclaBV[0], &tecla, sizeof(char));

      switch(tecla) {
            case (char)SDLK_LEFT:
                  posBV.x = (posBV.x > BV_RADIO) ? posBV.x-10 : 0;
                  shmBolaVerde[BV_X] = posBV.x;
            break;
            case (char)SDLK_RIGHT:
                  posBV.x = (posBV.x < PANTALLA_ANCHO-BV_RADIO*2) ? posBV.x+10 :
PANTALLA_ANCHO-BV_RADIO*2;
                  shmBolaVerde[BV_X] = posBV.x;
            break;
            case (char)SDLK_UP:
                  posBV.y = (posBV.y > BV_RADIO) ? posBV.y-10 : 0;
                  shmBolaVerde[BV_Y] = posBV.y;
            break;
            case (char)SDLK_DOWN:
                  posBV.y = (posBV.y < PANTALLA_ALTO-BV_RADIO*2) ? posBV.y+10 :
PANTALLA_ALTO-BV_RADIO*2;
                  shmBolaVerde[BV_Y] = posBV.y;
            break;
            case (char)SDLK_RETURN:
                  bool visible;


                  kill(pidPrincipal, SIGLEERAZUL);
                  read(pipeIsVisibleBA[0], &visible, 1);
                  if( visible ) {
                        kill(pidPrincipal, SIGCOMPEQUENA);
                  }
            break;
      }
}

void dibujarBolaVerde(int x, int y) {
      SDL_Rect rect;

      rect = (SDL_Rect) { (Sint16)x, (Sint16)y, BV_RADIO*2, BV_RADIO*2 };
      SDL_BlitSurface(srfBolaVerde, NULL, pantalla, &rect);
}

int crearProcesoBolaPequena(int id, char dir) {
      pid_t pid;

      pid = fork();
```

```
	if( pid == 0 ) {
		printf("Bola Pequena iniciado (%d)...\n", getpid());
		int aumento;
		int x;



		if( dir == 'I' ) {
			aumento = -1;
		} else {
			aumento = 1;
		}



		x = shmBolaPequena[BP_X + id * shmTamanoBolaPequena];

		do {
			while(shmJuego[JUEGO_PAUSADO] && !shmJuego[JUEGO_FINAL]);

			x += aumento;
			shmBolaPequena[BP_X + id * shmTamanoBolaPequena] = x;
			SDL_Delay(1);
		} while(x >= -BP_RADIO*2 && x <= PANTALLA_ANCHO &&
shmJuego[JUEGO_FINAL] == 0);
		kill(pidPrincipal, SIGDESTRUIRPEQUENA);
		printf("Bola Pequena finalizado...\n");
		quick_exit(0);
	} else if( pid == -1 ) {
		perror("Error al crear el proceso Bolas Pequenas.");
		exit(1);
	}

	return pid;
}

void sigComPequena(int signal) {


	if( bpTotal < NUM_BOLAS_PEQUENAS ) {
		pid_t pidBP;
		int id;

		id = obtenerNuevoIdPequena();

		if( id != -1 ) {
			int idRoja = bolaRojaMasCercana();
			char dir;



			if( idRoja != -1 ) {
				if( shmBolaRoja[BR_X + idRoja * shmTamanoBolaRoja] <=
shmBolaVerde[BV_X] ) {
					dir = 'I';
				} else {
					dir = 'D';
				}
```

```
                } else {
                        dir = 'I';
                }


                shmBolaPequena[BP_X + id * shmTamanoBolaPequena] =
shmBolaVerde[BV_X] - BP_RADIO * 2;
                shmBolaPequena[BP_Y + id * shmTamanoBolaPequena] =
shmBolaVerde[BV_Y] + BV_RADIO - BP_RADIO;


                pidBP = crearProcesoBolaPequena(id, dir);
                bolasPequenas[id].pid = pidBP;
                bpTotal++;
            }
        }
}

void sigDestruirPequena(int signal, siginfo_t *info, void *data) {
        bool encontrado = false;

        bpTotal--;

        for(int i=0; i < NUM_BOLAS_PEQUENAS && !encontrado; i++) {
                if( bolasPequenas[i].pid == info->si_pid ) {
                        bolasPequenas[i].pid = 0;
                }
        }
}




int obtenerNuevoIdPequena() {
        bpUltimoId++;


        if( bpUltimoId >= NUM_BOLAS_PEQUENAS ) {
                bpUltimoId = 0;
        }



        if( bolasPequenas[bpUltimoId].pid ) {
                bool encontrado = false;

                for(int i=0; i < NUM_BOLAS_PEQUENAS && !encontrado; i++) {
                        bpUltimoId++;

                        if( bpUltimoId >= NUM_BOLAS_PEQUENAS ) bpUltimoId = 0;



                        if( !bolasPequenas[bpUltimoId].pid ) {
                                encontrado = true;
                        }
                }


                if( !encontrado ) return -1;
```

```
	}

	return bpUltimoId;
}

void registerSigPrincipalPequena() {

	struct sigaction act;
	act.sa_handler = &sigComPequena;
	act.sa_flags = SA_RESTART;
	sigemptyset(&act.sa_mask);
	sigaction(SIGPRINCIPALPEQUENA, &act, NULL);
}

void dibujarBolaPequena() {
	SDL_Rect rect;

	for(int i=0; i < NUM_BOLAS_PEQUENAS; i++) {
		if( bolasPequenas[i].pid ) {
			rect = (SDL_Rect) { (Sint16)shmBolaPequena[BP_X +
i*shmTamanoBolaPequena], (Sint16)shmBolaPequena[BP_Y + i*shmTamanoBolaPequena],
BP_RADIO*2, BP_RADIO*2 };
			SDL_BlitSurface(srfBolaPequena, NULL, pantalla, &rect);
		}
	}
}



int crearProcesoBolaRoja() {
	pid_t	pid;

	pid = fork();

	if( pid == 0 ) {
		Uint32			inicio,
					inicioCrecimiento[NUM_BOLAS_ROJAS];
		BolaRoja		br[NUM_BOLAS_ROJAS];
		int			nextId = 0;

		for(int i=0; i < NUM_BOLAS_ROJAS; i++) {
			inicioCrecimiento[i] = 0;
			timerIdRoja[i] = (SDL_TimerID)-1;
			inicializarBolaRoja(i, br);
		}


		if (SDL_Init(SDL_INIT_TIMER) == -1) {
			printf("No se pudo iniciar timer: %s\n", SDL_GetError());
			exit(1);
		}



		SDL_Delay(2000);

		printf("Bola Roja iniciado...\n");


		inicio = SDL_GetTicks();
		shmBolaRoja[BR_ACTIVA] = 1;
```

```
            timerIdRoja[0] = SDL_AddTimer(20, moverBolaRoja, &br[0]);
            inicioCrecimiento[0] = SDL_GetTicks();

            while(!shmJuego[JUEGO_FINAL]) {

                  while(shmJuego[JUEGO_PAUSADO] && !shmJuego[JUEGO_FINAL]) {
                        inicio = SDL_GetTicks();
                        for(int i=0; i < NUM_BOLAS_ROJAS; i++) {
                              inicioCrecimiento[i] = SDL_GetTicks();
                        }
                  }

                  if( SDL_GetTicks() - inicio >= (Uint32)(INTERVALO_ROJAS *
1000) ) {

                        nextId = getIdLibreRoja();
                        if( nextId != -1 ) {
                              inicializarBolaRoja(nextId, br);
                              shmBolaRoja[BR_ACTIVA + nextId*shmTamanoBolaRoja]
= 1;
                              timerIdRoja[nextId] = SDL_AddTimer(20,
moverBolaRoja, &br[nextId]);
                              inicioCrecimiento[nextId] = SDL_GetTicks();
                        }
                        inicio = SDL_GetTicks();
                  }


                  for(int i=0; i < NUM_BOLAS_ROJAS; i++) {
                        if( inicioCrecimiento[i] != 0 && SDL_GetTicks() -
inicioCrecimiento[i] >= TIEMPO_CRECIMIENTO * 1000 ) {
                              if( br[i].radio == BR_RADIO_INICIAL ) {
                                    br[i].radio = BR_RADIO_T2;
                              } else if( br[i].radio == BR_RADIO_T2 ) {
                                    br[i].radio = BR_RADIO_T3;
                              } else if( br[i].radio == BR_RADIO_T3 ) {
                                    br[i].radio = BR_RADIO_T4;
                              }

                              shmBolaRoja[BR_RADIO + i*shmTamanoBolaRoja] =
br[i].radio;

                              inicioCrecimiento[i] = SDL_GetTicks();
                        }
                  }
            }


            for(int i=0; i < NUM_BOLAS_ROJAS; i++) {
                  if( timerIdRoja[i] != (SDL_TimerID)-1 )
SDL_RemoveTimer(timerIdRoja[i]);
            }

            printf("Bola Roja finalizado...\n");
            exit(0);
      } else if( pid == -1 ) {
            perror("Error al crear el proceso Bola Roja.");
            exit(1);
      }

      return pid;
```

```
}


int getIdLibreRoja() {
	bool encontrado = false;
	int id = -1;

	for(int i=0; i < NUM_BOLAS_ROJAS && !encontrado; i++) {
		if( timerIdRoja[i] == (SDL_TimerID)-1 ) {
			id = i;
			encontrado = true;
		}
	}

	return id;
}

void inicializarBolaRoja(int id, BolaRoja *br) {
	br[id].id = id;
	br[id].pos.x = -BR_RADIO_INICIAL*2;
	br[id].pos.y = -BR_RADIO_INICIAL*2;
	br[id].radio = BR_RADIO_INICIAL;
	br[id].direccion = BR_DIR_DCHA_ABAJO;
	shmBolaRoja[BR_X + id*shmTamanoBolaRoja] = br[id].pos.x;
	shmBolaRoja[BR_Y + id*shmTamanoBolaRoja] = br[id].pos.y;
	shmBolaRoja[BR_RADIO + id*shmTamanoBolaRoja] = BR_RADIO_INICIAL;
	shmBolaRoja[BR_ACTIVA + id*shmTamanoBolaRoja] = 0;
}


Uint32 moverBolaRoja(Uint32 interval, void *param) {
	if( !shmJuego[JUEGO_PAUSADO] ) {
		BolaRoja *br = (BolaRoja *)param;
		int desplazamiento = br->id * shmTamanoBolaRoja;

		if( shmBolaRoja[BR_ACTIVA + desplazamiento] ) {

			switch(br->direccion) {
				case BR_DIR_IZQ_ARRIBA:
					br->pos.x = (br->pos.x > BR_VELOCIDAD) ? br-
>pos.x-BR_VELOCIDAD : 0;
					br->pos.y = (br->pos.y > BR_VELOCIDAD) ? br-
>pos.y-BR_VELOCIDAD : 0;


					if( br->pos.x == 0 && br->pos.y == 0 ) {
						br->direccion = BR_DIR_DCHA_ABAJO;

					} else if( br->pos.x == 0 && br->pos.y != 0 ) {
						br->direccion = BR_DIR_DCHA_ARRIBA;

					} else if( br->pos.x != 0 && br->pos.y == 0 ) {
						br->direccion = BR_DIR_IZQ_ABAJO;
					}
				break;
				case BR_DIR_IZQ_ABAJO:
					br->pos.x = (br->pos.x > BR_VELOCIDAD) ? br-
>pos.x-BR_VELOCIDAD : 0;
					br->pos.y = (br->pos.y < PANTALLA_ALTO-
BR_VELOCIDAD-br->radio*2) ? br->pos.y+BR_VELOCIDAD : PANTALLA_ALTO-br->radio*2;
```

```
                                if( br->pos.x == 0 && br->pos.y == PANTALLA_ALTO-
br->radio*2 ) {
                                      br->direccion = BR_DIR_DCHA_ARRIBA;

                                } else if( br->pos.x == 0 && br->pos.y !=
PANTALLA_ALTO-br->radio*2 ) {
                                      br->direccion = BR_DIR_DCHA_ABAJO;

                                } else if( br->pos.x != 0 && br->pos.y ==
PANTALLA_ALTO-br->radio*2 ) {
                                      br->direccion = BR_DIR_IZQ_ARRIBA;
                                }
                          break;
                          case BR_DIR_DCHA_ARRIBA:
                                br->pos.x = (br->pos.x < PANTALLA_ANCHO-
BR_VELOCIDAD-br->radio*2) ? br->pos.x+BR_VELOCIDAD : PANTALLA_ANCHO-br-
>radio*2;
                                br->pos.y = (br->pos.y > BR_VELOCIDAD) ? br-
>pos.y-BR_VELOCIDAD : 0;


                                if( br->pos.x == PANTALLA_ANCHO-br->radio*2 && br-
>pos.y == 0 ) {
                                      br->direccion = BR_DIR_IZQ_ABAJO;

                                } else if( br->pos.x == PANTALLA_ANCHO-br->radio*2
&& br->pos.y != 0 ) {
                                      br->direccion = BR_DIR_IZQ_ARRIBA;

                                } else if( br->pos.x != PANTALLA_ANCHO-br->radio*2
&& br->pos.y == 0 ) {
                                      br->direccion = BR_DIR_DCHA_ABAJO;
                                }
                          break;
                          case BR_DIR_DCHA_ABAJO:
                                br->pos.x = (br->pos.x < PANTALLA_ANCHO-
BR_VELOCIDAD-br->radio*2) ? br->pos.x+BR_VELOCIDAD : PANTALLA_ANCHO-br-
>radio*2;
                                br->pos.y = (br->pos.y < PANTALLA_ALTO-
BR_VELOCIDAD-br->radio*2) ? br->pos.y+BR_VELOCIDAD : PANTALLA_ALTO-br->radio*2;


                                if( br->pos.x == PANTALLA_ANCHO-br->radio*2 && br-
>pos.y == PANTALLA_ALTO-br->radio*2 ) {
                                      br->direccion = BR_DIR_IZQ_ARRIBA;

                                } else if( br->pos.x == PANTALLA_ANCHO-br->radio*2
&& br->pos.y != PANTALLA_ALTO-br->radio*2 ) {
                                      br->direccion = BR_DIR_IZQ_ABAJO;

                                } else if( br->pos.x != PANTALLA_ANCHO-br->radio*2
&& br->pos.y == PANTALLA_ALTO-br->radio*2 ) {
                                      br->direccion = BR_DIR_DCHA_ARRIBA;
                                }
                          break;
                    }


                    shmBolaRoja[BR_X+desplazamiento] = br->pos.x;
                    shmBolaRoja[BR_Y+desplazamiento] = br->pos.y;
```

```
            } else {

                  SDL_RemoveTimer(timerIdRoja[br->id]);
                  timerIdRoja[br->id] = (SDL_TimerID)-1;
            }
      }

      return interval;
}

void dibujarBolaRoja() {
      SDL_Rect    rect,
                        srcRect;
      int desplazamiento,
            x,
            y,
            srcX,
            radio;


      for(int i=0; i < NUM_BOLAS_ROJAS; i++) {
            desplazamiento = i*shmTamanoBolaRoja;
            if( shmBolaRoja[BR_ACTIVA + desplazamiento] ) {
                  x = shmBolaRoja[BR_X+desplazamiento];
                  y = shmBolaRoja[BR_Y+desplazamiento];
                  radio = shmBolaRoja[BR_RADIO+desplazamiento];

                  if( radio == BR_RADIO_INICIAL ) {
                        srcX = BR_POS_1;
                  } else if( radio == BR_RADIO_T2 ) {
                        srcX = BR_POS_2;
                  } else if ( radio == BR_RADIO_T3 ) {
                        srcX = BR_POS_3;
                  } else if( radio == BR_RADIO_T4 ) {
                        srcX = BR_POS_4;
                  }

                  srcRect = (SDL_Rect) { (Sint16)srcX, (Sint16)0, (Uint16)
(radio*2), (Uint16)(radio*2) };
                  rect = (SDL_Rect) { (Sint16)x, (Sint16)y, (Uint16)(radio*2),
(Uint16)(radio*2) };
                  SDL_BlitSurface(srfBolaRoja, &srcRect, pantalla, &rect);
            }
      }
}

int crearProcesoBolaAzul() {
      pid_t pid;

      if( pipe(pipePosBA) == -1 ) {
            printf("Error al intentar crear el pipe pipePosBA.\n");
      }

      pid = fork();

      if( pid == 0 ) {
            printf("Bola Azul iniciado...\n");
            close(pipePosBA[0]);

            srand(time(NULL));
```

```
            Point pos;
            pos.x = rand() % (PANTALLA_ANCHO - BA_RADIO*2);
            pos.y = rand() % (PANTALLA_ALTO - BA_RADIO*2);



            write(pipePosBA[1], &pos, sizeof(Point));
            kill(pidPrincipal, SIGMOVAZUL);
            SDL_Delay(20);
            kill(pidPrincipal, SIGTOGGLEAZUL);

            do {
                  while(shmJuego[JUEGO_PAUSADO] && !shmJuego[JUEGO_FINAL]) {


                  }



                  pos.y++;
                  write(pipePosBA[1], &pos, sizeof(Point));
                  SDL_Delay(20);
                  kill(pidPrincipal, SIGMOVAZUL);
            } while(pos.y <= PANTALLA_ALTO && !shmJuego[JUEGO_FINAL]);



            kill(pidPrincipal, SIGTOGGLEAZUL);
            printf("Bola Azul finalizado...\n");
            quick_exit(0);
      } else if( pid == -1 ) {
            perror("Error al crear el proceso Bola Azul.");
            exit(1);
      }

      close(pipePosBA[1]);
      return pid;
}

void sigBA(int signal) {
      read(pipePosBA[0], &posAzul, sizeof(Point));
      dibujarBolaAzul();
}

void sigToggleBA(int signal) {
      isVisibleAzul = !isVisibleAzul;
}

void sigLeerBA(int signal) {
      write(pipeIsVisibleBA[1], &isVisibleAzul, 1);
}

void dibujarBolaAzul() {
      SDL_Rect rect;

      rect = (SDL_Rect) { (Sint16)posAzul.x, (Sint16)posAzul.y, (Uint16)
(BA_RADIO*2), (Uint16)(BA_RADIO*2) };
      SDL_BlitSurface(srfBolaAzul, NULL, pantalla, &rect);
}
```

```
int crearProcesoBolaNegra() {
	pid_t	pid;

	pid = fork();

	if( pid == 0 ) {
		Uint32			inicio = 0;
		BolaNegra		bn[NUM_BOLAS_NEGRAS];
		int			nextId = 0;


		for(int i=0; i < NUM_BOLAS_NEGRAS; i++) {
			timerIdNegra[i] = (SDL_TimerID)-1;
			inicializarBolaNegra(i, bn);
		}


		if (SDL_Init(SDL_INIT_TIMER) == -1) {
			printf("No se pudo iniciar timer: %s\n", SDL_GetError());
			exit(1);
		}



		SDL_Delay(2000);

		inicio = SDL_GetTicks();

		printf("Bola Negra iniciado...\n");

		while(!shmJuego[JUEGO_FINAL]) {
			while(shmJuego[JUEGO_PAUSADO] && !shmJuego[JUEGO_FINAL]) {
				inicio = SDL_GetTicks();
			}



			if( SDL_GetTicks() - inicio >= (Uint32)(INTERVALO_NEGRAS *
1000) ) {
				nextId = getIdLibreNegra();
				if( nextId != -1 ) {
					inicializarBolaNegra(nextId, bn);
					shmBolaNegra[BN_ACTIVA +
nextId*shmTamanoBolaNegra] = 1;
					timerIdNegra[nextId] = SDL_AddTimer(20,
moverBolaNegra, &bn[nextId]);
				}
				inicio = SDL_GetTicks();
			}
		}


		for(int i=0; i < NUM_BOLAS_NEGRAS; i++) {
			if( timerIdNegra[i] != (SDL_TimerID)-1 )
SDL_RemoveTimer(timerIdNegra[i]);
		}

		printf("Bola Negra finalizado...\n");
		exit(0);
	} else if( pid == -1 ) {
		perror("Error al crear el proceso Bola Negra.");
```

```
            exit(1);
      }

      return pid;
}


int getIdLibreNegra() {
      bool encontrado = false;
      int id = -1;

      for(int i=0; i < NUM_BOLAS_NEGRAS && !encontrado; i++) {
            if( timerIdNegra[i] == (SDL_TimerID)-1 ) {
                  id = i;
                  encontrado = true;
            }
      }

      return id;
}

void inicializarBolaNegra(int id, BolaNegra *bn) {

      srand(time(NULL));

      bn[id].id = id;
      bn[id].pos.x = rand() % (PANTALLA_ANCHO-BN_RADIO);
      bn[id].pos.y = rand() % (PANTALLA_ALTO-BN_RADIO);
      bn[id].direccion = rand() % 4;
      shmBolaNegra[BN_X + id*shmTamanoBolaNegra] = bn[id].pos.x;
      shmBolaNegra[BN_Y + id*shmTamanoBolaNegra] = bn[id].pos.y;
      shmBolaNegra[BN_ACTIVA + id*shmTamanoBolaNegra] = 0;
}


Uint32 moverBolaNegra(Uint32 interval, void *param) {
      if( !shmJuego[JUEGO_PAUSADO] ) {
            BolaNegra *bn = (BolaNegra *)param;
            int desplazamiento = bn->id * shmTamanoBolaNegra;

            if( shmBolaNegra[BN_ACTIVA + desplazamiento] ) {

                  switch(bn->direccion) {
                        case BN_DIR_IZQ_ARRIBA:
                              bn->pos.x = (bn->pos.x > BN_VELOCIDAD) ? bn-
>pos.x-BN_VELOCIDAD : 0;
                              bn->pos.y = (bn->pos.y > BN_VELOCIDAD) ? bn-
>pos.y-BN_VELOCIDAD : 0;


                              if( bn->pos.x == 0 ) {
                                    shmBolaNegra[BN_ACTIVA+desplazamiento] = 0;
                              } else if( bn->pos.x != 0 && bn->pos.y == 0 ) {
                                    bn->direccion = BN_DIR_IZQ_ABAJO;
                              }
                        break;
                        case BN_DIR_IZQ_ABAJO:
                              bn->pos.x = (bn->pos.x > BN_VELOCIDAD) ? bn-
>pos.x-BN_VELOCIDAD : 0;
                              bn->pos.y = (bn->pos.y < PANTALLA_ALTO-
BN_VELOCIDAD-BN_RADIO*2) ? bn->pos.y+BN_VELOCIDAD : PANTALLA_ALTO-BN_RADIO*2;
```

```
                              if( bn->pos.x == 0 ) {
                                    shmBolaNegra[BN_ACTIVA+desplazamiento] = 0;
                              } else if( bn->pos.x != 0 && bn->pos.y ==
PANTALLA_ALTO-BN_RADIO*2 ) {
                                    bn->direccion = BN_DIR_IZQ_ARRIBA;
                              }
                        break;
                        case BN_DIR_DCHA_ARRIBA:
                              bn->pos.x = (bn->pos.x < PANTALLA_ANCHO-
BN_VELOCIDAD-BN_RADIO*2) ? bn->pos.x+BN_VELOCIDAD : PANTALLA_ANCHO-BN_RADIO*2;
                              bn->pos.y = (bn->pos.y > BN_VELOCIDAD) ? bn-
>pos.y-BN_VELOCIDAD : 0;


                              if( bn->pos.x == PANTALLA_ANCHO-BN_RADIO*2 ) {
                                    shmBolaNegra[BN_ACTIVA+desplazamiento] = 0;
                              } else if( bn->pos.x != PANTALLA_ANCHO-BN_RADIO*2
&& bn->pos.y == 0 ) {
                                    bn->direccion = BN_DIR_DCHA_ABAJO;
                              }
                        break;
                        case BN_DIR_DCHA_ABAJO:
                              bn->pos.x = (bn->pos.x < PANTALLA_ANCHO-
BN_VELOCIDAD-BN_RADIO*2) ? bn->pos.x+BN_VELOCIDAD : PANTALLA_ANCHO-BN_RADIO*2;
                              bn->pos.y = (bn->pos.y < PANTALLA_ALTO-
BN_VELOCIDAD-BN_RADIO*2) ? bn->pos.y+BN_VELOCIDAD : PANTALLA_ALTO-BN_RADIO*2;


                              if( bn->pos.x == PANTALLA_ANCHO-BN_RADIO*2 ) {
                                    shmBolaNegra[BN_ACTIVA+desplazamiento] = 0;
                              } else if( bn->pos.x != PANTALLA_ANCHO-BN_RADIO*2
&& bn->pos.y == PANTALLA_ALTO-BN_RADIO*2 ) {
                                    bn->direccion = BN_DIR_DCHA_ARRIBA;
                              }
                        break;
                  }


                  shmBolaNegra[BN_X+desplazamiento] = bn->pos.x;
                  shmBolaNegra[BN_Y+desplazamiento] = bn->pos.y;
            } else {

                  SDL_RemoveTimer(timerIdNegra[bn->id]);
                  timerIdNegra[bn->id] = (SDL_TimerID)-1;
            }
      }

      return interval;
}

void dibujarBolaNegra() {
      SDL_Rect rect;
      int desplazamiento,
            x,
            y;


      for(int i=0; i < NUM_BOLAS_NEGRAS; i++) {
            desplazamiento = i*shmTamanoBolaNegra;
```

```
            if( shmBolaNegra[BN_ACTIVA + desplazamiento] ) {
                  x = shmBolaNegra[BN_X+desplazamiento];
                  y = shmBolaNegra[BN_Y+desplazamiento];

                  rect = (SDL_Rect) { (Sint16)x, (Sint16)y, (Uint16)
(BN_RADIO*2), (Uint16)(BN_RADIO*2) };
                  SDL_BlitSurface(srfBolaNegra, NULL, pantalla, &rect);
            }
      }
}

void inicializar() {

      if( SDL_Init(SDL_INIT_VIDEO) == -1 ) {
            printf("No se pudo iniciar el video: %s\n", SDL_GetError());
            exit(1);
      }

      if( SDL_Init(SDL_INIT_TIMER) == -1 ) {
            printf("No se pudo iniciar el timer: %s\n", SDL_GetError());
            exit(1);
      }

      if( Mix_OpenAudio( 22050, MIX_DEFAULT_FORMAT, 2, 4096 ) == -1 ) {
        printf("No se pudo iniciar el audio: %s\n", SDL_GetError());
    }

    if( TTF_Init() == -1 ) {
        printf("No se pudo iniciar True Type Font: %s\n", SDL_GetError());
            exit(1);
    }

      pantalla = SDL_SetVideoMode(PANTALLA_ANCHO, PANTALLA_ALTO, 32,
SDL_HWSURFACE | SDL_DOUBLEBUF);
      if(!pantalla) {
            printf("No se pudo iniciar el modo de pantalla: %s\n",
SDL_GetError());
            exit(1);
      }

      fondo = IMG_Load("img/bg.jpg");
      if( !fondo ) {
            printf("No se pudo cargar el fondo.\n");
            exit(1);
      }

      srfFlash = IMG_Load("img/flash.png");
      if( !srfFlash ) {
            printf("No se pudo cargar la imagen del flash.\n");
            exit(1);
      }

      srfEnPausa = IMG_Load("img/pausa.png");
      if( !srfEnPausa ) {
            printf("No se pudo cargar la imagen de pausa.\n");
            exit(1);
      }

      srfHasPerdido = IMG_Load("img/hasperdido.png");
      if( !srfHasPerdido ) {
            printf("No se pudo cargar la imagen que se muestra cuando
```

```
pierdes.\n");
            exit(1);
      }

      srfVolumen = IMG_Load("img/volumen.png");
      if( !srfVolumen ) {
            printf("No se pudo cargar la imagen del volumen.\n");
            exit(1);
      }

      srfBolaVerde = IMG_Load("img/bolaverde.png");
      if( !srfBolaVerde ) {
            printf("No se pudo cargar la imagen de la bola verde.\n");
            exit(1);
      }

      srfBolaRoja = IMG_Load("img/bolaroja.png");
      if( !srfBolaRoja ) {
            printf("No se pudo cargar la imagen de la bola roja.\n");
            exit(1);
      }

      srfBolaPequena = IMG_Load("img/bolapequena.png");
      if( !srfBolaPequena ) {
            printf("No se pudo cargar la imagen de la bola pequeña.\n");
            exit(1);
      }

      srfBolaAzul = IMG_Load("img/bolaazul.png");
      if( !srfBolaAzul ) {
            printf("No se pudo cargar la imagen de la bola azul.\n");
            exit(1);
      }

      srfBolaNegra = IMG_Load("img/bolanegra.png");
      if( !srfBolaNegra ) {
            printf("No se pudo cargar la imagen de la bola negra.\n");
            exit(1);
      }

      colorBlanco = (SDL_Color){ 255, 255, 255 };
      colorRojo = (SDL_Color){ 255, 0, 0 };

      fontPuntuacion = TTF_OpenFont("fonts/american-captain.ttf", 28);
      if( fontPuntuacion == NULL ) {
            printf("No se pudo cargar la fuente American Captain.\n");
            exit(1);
      }

    musica = Mix_LoadMUS("snd/dnbweird.ogg");
    if( musica == NULL ) {
      printf("No se pudo cargar la cancion dnbweird.ogg.\n");
    }

    Mix_VolumeMusic(MIX_MAX_VOLUME / 2);
    Mix_Volume(-1, MIX_MAX_VOLUME / 2);

    if( Mix_PlayMusic(musica, -1) == -1 ) {
            printf("No se ha podido reproducir la musica.\n");
      }
```

```
	shot = Mix_LoadWAV("snd/shot.wav");
	explosion = Mix_LoadWAV("snd/explosion.wav");
	whip = Mix_LoadWAV("snd/whip.wav");


	SDL_WM_SetCaption( "Orb Attack por Francisco Mendez Vilas", NULL );

	SDL_EnableKeyRepeat(200, 0);

	shmJuego[JUEGO_FINAL] = 0;
	shmJuego[JUEGO_PAUSADO] = 0;
	shmJuego[JUEGO_PUNTOS] = ENERGIA_BOLA_VERDE;
}


void cargarConfiguracion() {
	ifstream fichero("config.txt");
	string linea;
	char  *clave;
	int   valor,
		contador = 0;



	while(contador < 13 && getline(fichero, linea)) {
	    clave = strtok((char *)linea.c_str(), " \t");
	      valor = atoi(strtok(NULL, " \t"));

	      if( strcmp(clave, "energia_bolas_verdes") == 0 ) {
		    ENERGIA_BOLA_VERDE = valor;
	      } else if( strcmp(clave, "num_bolas_rojas") == 0 ) {
		    NUM_BOLAS_ROJAS = valor;
	      } else if( strcmp(clave, "intervalo_rojas") == 0 ) {
		    INTERVALO_ROJAS = valor;
	      } else if( strcmp(clave, "intervalo_azules") == 0 ) {
		    INTERVALO_AZUL = valor;
	      } else if( strcmp(clave, "num_bolas_negras") == 0 ) {
		    NUM_BOLAS_NEGRAS = valor;
	      } else if( strcmp(clave, "intervalo_negras") == 0 ) {
		    INTERVALO_NEGRAS = valor;
	      } else if( strcmp(clave, "bolas_verdes_pequeñas") == 0 ) {
		    NUM_BOLAS_PEQUENAS = valor;
	      } else if( strcmp(clave, "puntuacion_t1_roja") == 0 ) {
		    PUNTUACION_T1_ROJA = valor;
	      } else if( strcmp(clave, "puntuacion_t2_roja") == 0 ) {
		    PUNTUACION_T2_ROJA = valor;
	      } else if( strcmp(clave, "puntuacion_t3_roja") == 0 ) {
		    PUNTUACION_T3_ROJA = valor;
	      } else if( strcmp(clave, "puntuacion_t4_roja") == 0 ) {
		    PUNTUACION_T4_ROJA = valor;
	      } else if( strcmp(clave, "puntuacion_bola_negra") == 0 ) {
		    PUNTUACION_NEGRA = valor;
	      } else if( strcmp(clave, "quita_energia_roja") == 0 ) {
		    QUITA_ENERGIA_ROJA = valor;
	      }

	      contador++;
	}
```

```
      timerIdRoja = (SDL_TimerID *)malloc(sizeof(SDL_TimerID) *
NUM_BOLAS_ROJAS);
      timerIdNegra = (SDL_TimerID *)malloc(sizeof(SDL_TimerID) *
NUM_BOLAS_NEGRAS);
}



void mostrarPantallaInicio() {
      bool continuar = false;
      SDL_Rect rect;
      SDL_Event event;
      SDL_Surface *imagen;

      if( SDL_Init(SDL_INIT_VIDEO) == -1 ) {
            printf("No se pudo iniciar el video: %s\n", SDL_GetError());
            exit(1);
      }

      if( Mix_OpenAudio( 22050, MIX_DEFAULT_FORMAT, 2, 4096 ) == -1 ) {
        printf("No se pudo iniciar el mezclador de Audio: %s\n",
SDL_GetError());
    }

    musicaInicio = Mix_LoadMUS("snd/funkyphresh.ogg");
    if( musicaInicio == NULL ) {
      printf("No se pudo cargar la cancion funkyphresh.ogg.\n");
    }

      SDL_WM_SetCaption( "Orb Attack por Francisco Mendez Vilas", NULL );

      pantalla = SDL_SetVideoMode(PANTALLA_ANCHO, PANTALLA_ALTO, 32,
SDL_HWSURFACE | SDL_DOUBLEBUF);
      if(!pantalla) {
            printf("No se pudo iniciar el modo de pantalla: %s\n",
SDL_GetError());
            exit(1);
      }

      srfInicio = IMG_Load("img/inicio.jpg");
      if( !srfInicio ) {
            printf("No se pudo cargar la imagen de inicio.\n");
            exit(1);
      }

      srfInstrucciones = IMG_Load("img/instrucciones.jpg");
      if( !srfInstrucciones ) {
            printf("No se pudo cargar la imagen de instrucciones.\n");
            exit(1);
      }

      imagen = srfInicio;

      Mix_VolumeMusic(MIX_MAX_VOLUME / 2);
      Mix_FadeInMusic(musicaInicio, -1, 2000);

      do {
            rect = (SDL_Rect) { 0, 0, PANTALLA_ANCHO, PANTALLA_ALTO };
            SDL_BlitSurface(imagen , NULL, pantalla, &rect);
```

```
            SDL_Flip(pantalla);

            SDL_WaitEvent(&event);
            if( event.type == SDL_KEYDOWN ) {
                  switch(event.key.keysym.sym) {
                        case SDLK_ESCAPE:
                              exit(0);
                        break;
                        case SDLK_RETURN:
                              continuar = true;
                        break;
                        case SDLK_i:
                              if( imagen == srfInicio ) {
                                    imagen = srfInstrucciones;
                              } else {
                                    imagen = srfInicio;
                              }
                        break;
                        default:
                        break;
                  }
            } else if( event.type == SDL_QUIT ) {
                  exit(0);
            }
      } while(!continuar);

      Mix_FadeOutMusic(1000);
      SDL_Delay(1000);

      SDL_FreeSurface(srfInicio);
      SDL_FreeSurface(srfInstrucciones);
      Mix_FreeMusic(musicaInicio);
      Mix_CloseAudio();
      SDL_Quit();
}

void dibujarFondo() {
      SDL_Rect rect;

      rect = (SDL_Rect) { 0, 0, PANTALLA_ANCHO, PANTALLA_ALTO };
      SDL_BlitSurface(fondo , NULL, pantalla, &rect);
}

void dibujarFlash() {
      SDL_Rect rect;

      rect = (SDL_Rect) { 0, 0, PANTALLA_ANCHO, PANTALLA_ALTO };
      SDL_BlitSurface(srfFlash , NULL, pantalla, &rect);
}

void dibujarEnPausa() {
      SDL_Rect rect;

      rect = (SDL_Rect) { 0, 0, PANTALLA_ANCHO, PANTALLA_ALTO };
      SDL_BlitSurface(srfEnPausa , NULL, pantalla, &rect);
}

void dibujarHasPerdido() {
      SDL_Rect rect;

      rect = (SDL_Rect) { 0, 0, PANTALLA_ANCHO, PANTALLA_ALTO };
```

```
	SDL_BlitSurface(srfHasPerdido , NULL, pantalla, &rect);
}

void dibujarPuntuacion() {
	SDL_Rect rect;
	char strPuntuacion[20];
	SDL_Color colorTexto;

	colorTexto = shmJuego[JUEGO_PUNTOS] < 200 ? colorRojo : colorBlanco;

	sprintf(strPuntuacion, "Puntos: %d", shmJuego[JUEGO_PUNTOS]);
	srfPuntuacion = TTF_RenderText_Blended(fontPuntuacion, strPuntuacion,
colorTexto);

	rect = (SDL_Rect) { (Sint16)10, (Sint16)10, PANTALLA_ANCHO-20, 30 };
	SDL_BlitSurface(srfPuntuacion, NULL, pantalla, &rect);
}

void dibujarVolumen() {
	SDL_Rect	rect,
			rect2;
	Sint16 altura;
	int volumen = Mix_VolumeMusic(-1);

	if( volumen == MIX_MAX_VOLUME ) {
		altura = 0;
	} else if( volumen >= (MIX_MAX_VOLUME / 4) * 3 ) {
		altura = 27;
	} else if( volumen >= MIX_MAX_VOLUME / 2 ) {
		altura = 54;
	} else if( volumen > 0 ) {
		altura = 81;
	} else {
		altura = 108;
	}

	rect = (SDL_Rect) { (Sint16)(PANTALLA_ANCHO-140), (Sint16)10, (Uint32)130,
(Uint32)25 };
	rect2 = (SDL_Rect) { (Sint16)0, altura, (Uint32)130, (Uint32)27 };
	SDL_BlitSurface(srfVolumen , &rect2, pantalla, &rect);
}



int bolaRojaMasCercana() {
	double	distancia = -1,
		aux;
	int	id = -1;



	for(int i=0; i < NUM_BOLAS_ROJAS; i++) {
		if( shmBolaRoja[BR_ACTIVA + i * shmTamanoBolaRoja] ) {
			aux = distanciaBolaVerdeBolaRoja(i);

			if( distancia != -1 && aux < distancia ) {
				id = i;
			} else if( distancia == -1 ) {
				distancia = aux;
				id = i;
			}
```

```
            }
      }

      return id;
}



bool detectarColisionBVBR() {
      double      distancia;
      int   radioRoja;
      bool  hayColision = false;

      for(int i=0; i < NUM_BOLAS_ROJAS; i++) {
            if( shmBolaRoja[BR_ACTIVA + i * shmTamanoBolaRoja] ) {
                  radioRoja = shmBolaRoja[BR_RADIO + i * shmTamanoBolaRoja];
                  distancia = distanciaBolaVerdeBolaRoja(i);

                  if( distancia <= radioRoja + BV_RADIO ) {
                        shmJuego[JUEGO_PUNTOS] -= QUITA_ENERGIA_ROJA;
                        hayColision = true;
                  }
            }
      }

      return hayColision;
}


double distanciaBolaVerdeBolaRoja(int idRoja) {
      Point p1,
                  p2;
      int   radioRoja,
                  desplazamientoRoja;
      double      distancia;

      desplazamientoRoja = shmTamanoBolaRoja * idRoja;
      radioRoja = shmBolaRoja[BR_RADIO + desplazamientoRoja];
      p1.x = shmBolaRoja[BR_X + desplazamientoRoja] + radioRoja;
      p1.y = shmBolaRoja[BR_Y + desplazamientoRoja] + radioRoja;
      p2.x = shmBolaVerde[BV_X] + BV_RADIO;
      p2.y = shmBolaVerde[BV_Y] + BV_RADIO;
      distancia = sqrt( pow(abs(p2.x - p1.x),2) + pow(abs(p2.y - p1.y),2) );

      return distancia;
}



void detectarColisionBPBR() {
      double      distancia;
      int   radioRoja;

      for(int iBP=0; iBP < NUM_BOLAS_PEQUENAS; iBP++) {
            if( bolasPequenas[iBP].pid ) {
                  for(int i=0; i < NUM_BOLAS_ROJAS; i++) {
                        if( shmBolaRoja[BR_ACTIVA + i * shmTamanoBolaRoja] ) {
                              radioRoja = shmBolaRoja[BR_RADIO + i *
shmTamanoBolaRoja];
                              distancia = distanciaBolaPequenaBolaRoja(iBP, i);
```

```
                              if( distancia <= radioRoja + BP_RADIO ) {
                                    kill(bolasPequenas[iBP].pid, SIGKILL);
                                    bolasPequenas[iBP].pid = 0;
                                    bpTotal--;
                                    shmBolaRoja[BR_ACTIVA + i *
shmTamanoBolaRoja] = 0;
                                    Mix_PlayChannel(-1, explosion, 0);



                                    switch(radioRoja) {
                                          case BR_RADIO_INICIAL:
                                                shmJuego[JUEGO_PUNTOS] +=
PUNTUACION_T1_ROJA;
                                          break;
                                          case BR_RADIO_T2:
                                                shmJuego[JUEGO_PUNTOS] +=
PUNTUACION_T2_ROJA;
                                          break;
                                          case BR_RADIO_T3:
                                                shmJuego[JUEGO_PUNTOS] +=
PUNTUACION_T3_ROJA;
                                          break;
                                          case BR_RADIO_T4:
                                                shmJuego[JUEGO_PUNTOS] +=
PUNTUACION_T4_ROJA;
                                          break;
                                    }
                              }
                        }
                  }
            }
      }
}



double distanciaBolaPequenaBolaRoja(int idBP, int idRoja) {
      Point p1,
                  p2;
      int   radioRoja,
                  desplazamiento;
      double      distancia;

      desplazamiento = shmTamanoBolaRoja * idRoja;
      radioRoja = shmBolaRoja[BR_RADIO + desplazamiento];
      p1.x = shmBolaRoja[BR_X + desplazamiento] + radioRoja;
      p1.y = shmBolaRoja[BR_Y + desplazamiento] + radioRoja;
      p2.x = shmBolaPequena[BP_X + idBP * shmTamanoBolaPequena] + BP_RADIO;
      p2.y = shmBolaPequena[BP_Y + idBP * shmTamanoBolaPequena] + BP_RADIO;
      distancia = sqrt( pow(abs(p2.x - p1.x),2) + pow(abs(p2.y - p1.y),2) );

      return distancia;
}



void detectarColisionBPBA() {
      if( isVisibleAzul ) {
            double distancia;
```

```
            Point p1,
                        p2;

            for(int i=0; i < NUM_BOLAS_PEQUENAS; i++) {
                  if( bolasPequenas[i].pid ) {
                        p1.x = posAzul.x + BA_RADIO;
                        p1.y = posAzul.y + BA_RADIO;
                        p2.x = shmBolaPequena[BP_X + i*shmTamanoBolaPequena] +
BP_RADIO;
                        p2.y = shmBolaPequena[BP_Y + i*shmTamanoBolaPequena] +
BP_RADIO;

                        distancia = sqrt( pow(abs(p2.x - p1.x),2) + pow(abs(p2.y
- p1.y),2) );

                        if( distancia <= BA_RADIO + BP_RADIO ) {
                              kill(bolasPequenas[i].pid, SIGKILL);
                              bolasPequenas[i].pid = 0;
                              bpTotal--;
                              if( pidBolaAzul ) kill(pidBolaAzul, SIGKILL);
                              pidBolaAzul = 0;
                              isVisibleAzul = false;
                              Mix_PlayChannel(-1, whip, 0);
                        }
                  }
            }
      }
}


int detectarColisionBNBR() {
      double      distancia;
      int   radioRoja;

      for(int i=0; i < NUM_BOLAS_NEGRAS; i++) {
            for(int h=0; h < NUM_BOLAS_ROJAS; h++) {
                  if( shmBolaRoja[BR_ACTIVA + h * shmTamanoBolaRoja]
                        && shmBolaNegra[BN_ACTIVA + i * shmTamanoBolaNegra] )
                  {
                        radioRoja = shmBolaRoja[BR_RADIO + h *
shmTamanoBolaRoja];
                        distancia = distanciaBolaNegraBolaRoja(i, h);

                        if( distancia <= radioRoja + BN_RADIO ) {
                              shmBolaNegra[BN_ACTIVA + i * shmTamanoBolaNegra] =
0;
                              shmJuego[JUEGO_PUNTOS] += PUNTUACION_NEGRA;

                              Mix_PlayChannel(-1, explosion, 0);
                              return h;
                        }
                  }
            }

      }

      return -1;
}
```

```
double distanciaBolaNegraBolaRoja(int idNegra, int idRoja) {
      Point p1,
                  p2;
      int   radioRoja,
                  desplazamientoRoja,
                  desplazamientoNegra;
      double      distancia;

      desplazamientoRoja = shmTamanoBolaRoja * idRoja;
      radioRoja = shmBolaRoja[BR_RADIO + desplazamientoRoja];
      p1.x = shmBolaRoja[BR_X + desplazamientoRoja] + radioRoja;
      p1.y = shmBolaRoja[BR_Y + desplazamientoRoja] + radioRoja;
      desplazamientoNegra = shmTamanoBolaNegra * idNegra;
      p2.x = shmBolaNegra[BN_X + desplazamientoNegra] + BN_RADIO;
      p2.y = shmBolaNegra[BN_Y + desplazamientoNegra] + BN_RADIO;
      distancia = sqrt( pow(abs(p2.x - p1.x),2) + pow(abs(p2.y - p1.y),2) );

      return distancia;
}


void liberar() {
      printf("\nTerminando (liberando recursos)...\n");
      shmdt((char *)shmJuego);
      shmctl(shmIdJuego, IPC_RMID, 0);
      shmdt((char *)shmBolaVerde);
      shmctl(shmIdBolaVerde, IPC_RMID, 0);
      shmdt((char *)shmBolaRoja);
      shmctl(shmIdBolaRoja, IPC_RMID, 0);
      shmdt((char *)shmBolaPequena);
      shmctl(shmIdBolaPequena, IPC_RMID, 0);
      shmdt((char *)shmBolaNegra);
      shmctl(shmIdBolaNegra, IPC_RMID, 0);
      SDL_FreeSurface(srfFlash);
      SDL_FreeSurface(fondo);
      SDL_FreeSurface(srfBolaVerde);
      SDL_FreeSurface(srfBolaRoja);
      SDL_FreeSurface(srfBolaPequena);
      SDL_FreeSurface(srfBolaAzul);
      SDL_FreeSurface(srfBolaNegra);
      SDL_FreeSurface(srfPuntuacion);
      SDL_FreeSurface(srfVolumen);
      SDL_FreeSurface(srfEnPausa);
      SDL_FreeSurface(srfHasPerdido);
    TTF_CloseFont(fontPuntuacion);
    Mix_FreeMusic(musica);
    Mix_FreeChunk(shot);
    Mix_FreeChunk(explosion);
    Mix_FreeChunk(whip);
    Mix_CloseAudio();
      SDL_Quit();
}
```

Bibliografía

Cplusplus.com - http://www.cplusplus.com/

Stack Overflow - http://stackoverflow.com/

The GNU C Library - http://www.gnu.org/software/libc/manual/html_node/