Instituto Tecnológico de Costa Rica

Escuela de Ingeniería en Computación

Campus Tecnológico Central de Cartago

[IC2001] Estructuras de Datos

Profesor: Ing. Víctor Garro Abarca

Manual Técnico

Video Juego Clásico

José Daniel Araya Ortega c.2022209303

Jocsan Adriel Pérez Coto c.2022437948

Entrega: 07/11/2022

II Semestre

Contenido

Estructuras de datos

Lista enlaza simple: Para trabajar de manera correcta los elementos del juego se tiene que aplicar diferentes listas enlazadas simples, dependiendo del elemento, que nos permite agregar o eliminar dinámicamente los valores necesarios cuando el programa lo requiera, gracias a que trabaja con punteros y variables anónimas. Además de que esta se genera a base structs, lo que nos permite generar nuestros elementos con los parámetros que necesitemos, generando los nodos de la lista.

Un ejemplo de cómo se ve una lista enlazada es el siguiente:

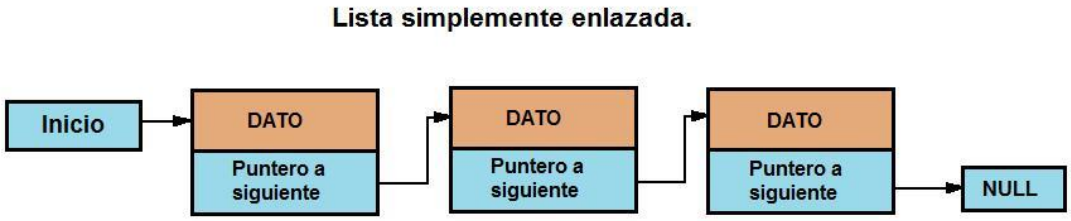


Fig 1. Lista enlazada simple. Tomado de: https://analisisyprogramacionoop.blogspot.com/2017/07/lista-simplemente-enlazada-C-sharp.html

Cola (FIFO): En este proyecto se implementará una cola de eventos, donde se debe primero encolar, y luego desencolar de manera dinámica los eventos. Esta se trabaja con un elemento específico de Allegro, que cumplirá con este propósito siendo que el programa esperará que eventos se encolen para poder trabajar desencolándolos.

Para el caso de la cola completa esta estará vacía y con forme se le encolan elementos, se le irán también desencolando otros. Para comprender este concepto en la real tenemos:

Fig 2. Ejemplo real de FIFO. Tomado de:
https://www.freepik.es/fotos-vectores-gratis/fila-de-personas

También lo podemos ver como una lista enlazada:

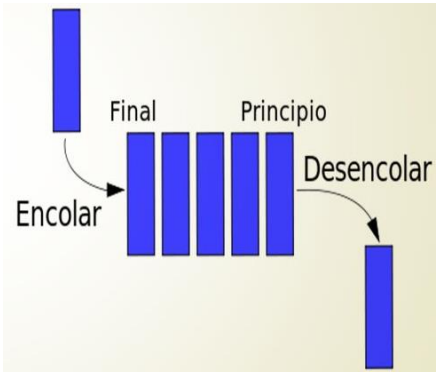


Fig 3. Ejemplo en LE de FIFO. Tomado de:
https://slideplayer.es/slide/16359303/

Pila (LIFO): Esta estructura se utilizará para el manejo de archivos, ya que se guardarán en forma de pila y al cargarlos se hará push, para siempre obtener los últimos en agregarse. La pila nos permite guardar elementos un metafórico compartimiento en orden y donde siempre el último en agregarse será el primero en salir.

Ejemplo:

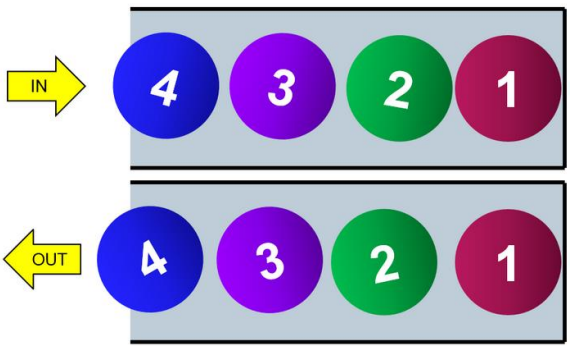


Fig 4. Ejemplo en LE de LIFO. Tomado de:
https://www.lawebdelprogramador.com/codigo/Python/6886-Clase-para-una-pila-LIFO-en-Python.html

Nodos

Para el proyecto se utilizarán 6 tipos nuevos de datos:

resultados: Nodo de lista enlazada resultados, el cual tiene array de chars para el nombre del jugador, el tiempo que duro, y los puntos obtenidos.

```
typedef struct resultados
{
    char Nombre[20];
    char Tiempo[20];
    char Puntaje[500];
    resultados* Siguiente;

}*PtrResultados;
```

mancha: Nodo de lista enlazada mancha, el cual tiene coordenadas x, y para saber dónde imprimir una mancha de sangre.

```
typedef struct mancha {
    int x;
    int y;
    mancha* Siguiente;
}*PtrMancha;
```

coche: Nodo de lista enlazada coche, este es el elemento del jugador.

```
typedef struct coche {
    int codigo;
    int SizeY;
    int SizeX;
    int x;
    int y;
    int velocidad;
    bool estado;
    int vida;
    int cont;
    int dano;
    coche* Siguiente;
}*PtrCoche;
```

Los parámetros son:

- **int código;**
- **int SizeY;** //Se refiere al alto.
- **int SizeX;** //Se refiere al ancho.

- **int x;** //Coordenada en eje x.
- **int y;** // Coordenada en eje y.
- **int velocidad;** //Velocidad del coche.
- **bool estado;** //Si está vivo o muerto, ósea activo o no.
- **int vida;** //Cantidad de vida del coche.
- **int cont;** //Contador para la animación de explosión.
- **int dano;** //Daño infringido al chocar.
- **coche* Siguiente;**

balaEsp: Nodo de lista enlazada balaEsp, estas será las balas que dispare el jugador.

```
typedef struct balaEsp {
    int x;
    int y;
    int SizeY;
    int SizeX;
    int velocidad;
    int direccion;
    int dano;
    bool estado;
    balaEsp* Siguiente;
}*PtrBalaEsp;
```

Los parámetros son:

- **int x;** //Coordenada en eje x.
- **int y;** // Coordenada en eje y.
- **int SizeY;** //Se refiere al alto.
- **int SizeX;** //Se refiere al ancho.
- **int velocidad;** //Velocidad de la bala.
- **int direccion;** //Dirección en la que se mueva la bala.
- **int dano;** //Daño hecho por la bala.
- **bool estado;** //Si está en el aire o no.
- **balaEsp* Siguiente;**

obsta: Nodo de lista enlazada obsta, estos serán los barriles que exploten al hacerles daño.

```
typedef struct obsta {
    int cont;
    int SizeY;
    int SizeX;
    int x;
    int y;
    bool estado;
    int dano;
    obsta* Siguiente;
}*PtrObsta;
```

Los parámetros son:

- **int cont;** //Contador para la animación de explosión, y de reaparición.
- **int SizeY;** //Se refiere al alto.
- **int SizeX;** //Se refiere al ancho.
- **int x;** //Coordenada en eje x.
- **int y;** // Coordenada en eje y.
- **bool estado;** //Si está en pantalla o no.
- **int dano;** //Daño que hacer al explotar con el coche.
- **obsta* Siguiente;**

Persona: Nodo de lista enlazada Persona, estos serán los objetivos a ser asesinados.

```
typedef struct Persona {
    int codigo;
    int cont;
    int SizeY;
    int SizeX;
    int x;
    int y;
    int vida;
    int dir;
    bool estado;
    int velocidad;
    Persona* Siguiente;
}*PtrPerso;
```

Los parámetros son:

int codigo; //Tipo de movimiento.

int cont; //Contador para la animación de sangre, y de reaparición.

int SizeY; //Se refiere al alto.

int SizeX; //Se refiere al ancho.

int x; //Coordenada en eje x.

int y; // Coordenada en eje y.

int vida; //Vida de la persona.

int dir; //Dirección del movimiento.

bool estado; //Si está vivo o muerto.

int velocidad; //Desplazamiento por tic.

Persona* Siguiente;

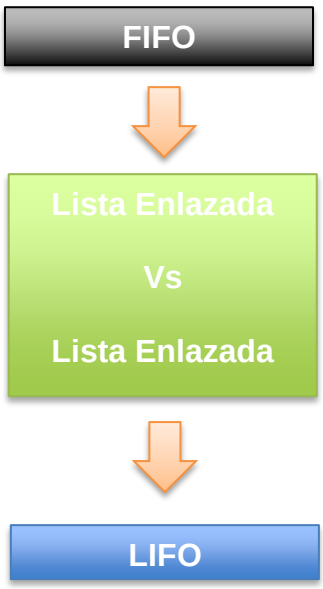
Relación entre estructuras

Primer tenemos que las listas enlazadas simples se relacionan mediante la búsqueda de colisiones, donde para cada lista se debe comparar con cada ciclo para determinar que si hay alguna interacción entre ellas.

A las anteriores se puede relacionar la lista FIFO, cola de eventos de allegro la cual es la que permite que se ejecuten estas comparaciones entre listas enlazadas, siendo que, aunque no se afectan de manera directa estas determinadas por la cola de eventos.

Además, podemos relacionar el uso de LIFO, con el hecho de que trabaja con archivos con los datos generados a partir de la interacción entre listas dado por la cola de eventos y los timers.

De esto podemos decir que tenemos una secuencia de interacciones como la siguiente:



Algoritmos/Lógica

RandomX: Dado una coordenada en X y un ancho genera un valor aleatorio que coincida dentro de un rango, para la coordenada X de un obstáculo.

```
int randomX(int R, int x) {
    int movR;
    bool ciclo = true;
    movR = 200 + rand() % (R - 330 - x);
    while (ciclo) {
        if (movR < (R - x) && movR > 200)
        return movR;
    }
}
```

RandomY: Dado una coordenada en Y y un alto genera un valor aleatorio que coincida dentro de un rango, para la coordenada Y de un obstáculo.

```
int randomY(int R, int y) {
    int movR;
    bool ciclo = true;
    movR = 90 + rand() % (R - 90 - y);
    while (ciclo) {
        if (movR < R - y && movR > 90)
        return movR;
    }
}
```

inicializarPlayer: Función que inicializa al jugador recibiendo la referencia de coche, las dimensiones de la pantalla y el bitmap del coche.

```
void inicializarPlayer(coche& jugador, int x, int y, ALLEGRO_BITMAP* map) {
    //Se crea la función inicializar jugador que ingresará os valores para c
    jugador.x = x / 2;
    jugador.y = y / 2;
    jugador.vida = 10;
    jugador.velocidad = 10;
    jugador.dano = 3;
    //Toma las medidas de alto y ancho del bitmap.
    jugador.SizeY = al_get_bitmap_height(bitmap: map);
    jugador.SizeX = al_get_bitmap_width(bitmap: map);
    jugador.estado = true;
    jugador.Siguiente = NULL;
    jugador.cont = 0;
}
```

colisionObjetos: Dado dos juegos de coordenada en x, coordenada en y, ancho y largo, devuelve true si detecta una colisión.

```
bool colisionObjetos(int x1, int y1, int w1, int h1, int x2, int y2, int w2, int h2) {
    if (x1 > x2 + w2 || x1 + w1 < x2 || y1 > y2 + h2 || y1 + h1 < y2) {
        return false;
    }
    else {
        return true;
    }
}
```

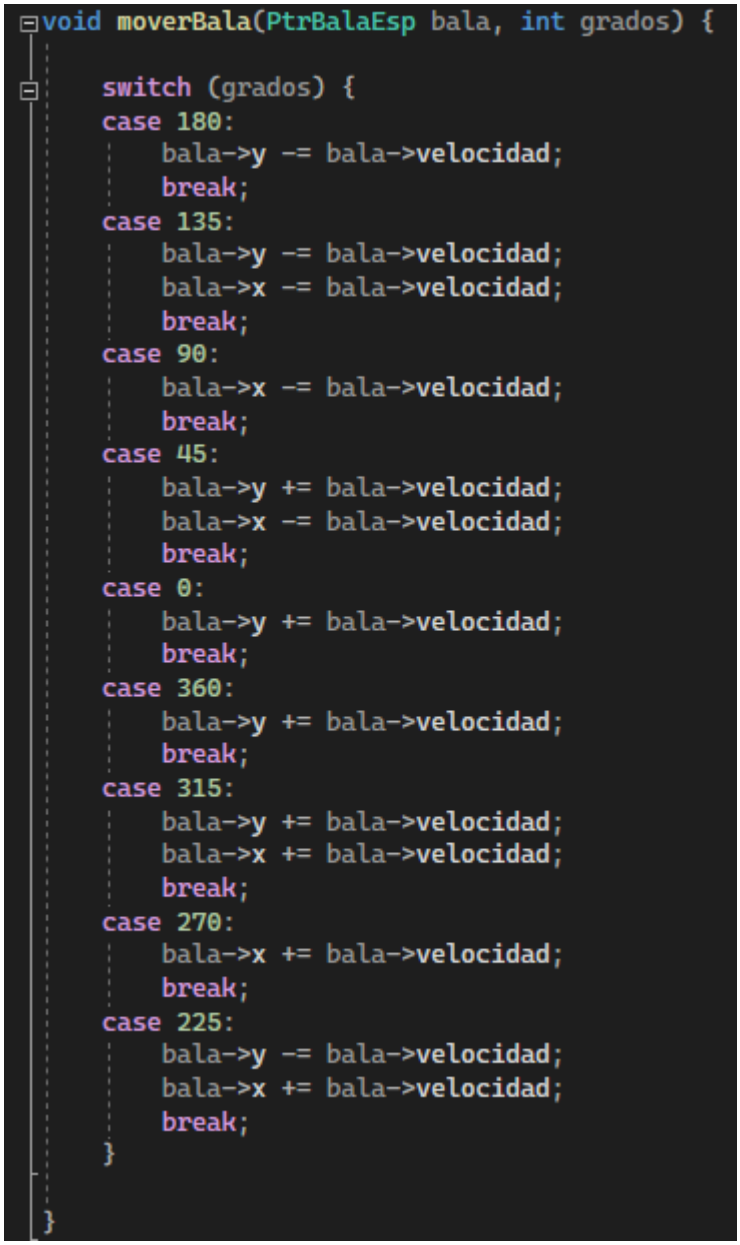
inicializarObs: Se inicializa un obstáculo igual que al jugador solo que se recibe una referencia obstáculo a iniciar y el coche como referencia para no generar obstáculos sobre él.

```
void inicializarObs(obsta& obs, int x, int y, ALLEGRO_BITMAP* map, coche player) {
    bool ciclo = true;
    while (ciclo) {
        obs.x = randomX(R: x, x: al_get_bitmap_width(bitmap: map));
        obs.y = randomY(R: y, y: al_get_bitmap_height(bitmap: map));
        //Si las coordenadas generadas hacen que halla colision con el area jugable pero no con el jugador, se mantienen.
        if (!colisionObjetos(x1: obs.x, y1: obs.y, w1: al_get_bitmap_width(bitmap: map), h1: al_get_bitmap_height(bitmap: map),
            x2: player.x - (player.SizeX / 2), y2: player.y - (player.SizeY / 2), w2: player.SizeX, h2: player.SizeY) &&
            colisionObjetos(x1: obs.x, y1: obs.y, w1: al_get_bitmap_width(bitmap: map), h1: al_get_bitmap_height(bitmap: map), x2: 200, y2: 90, w2: x - 130, h2: y))
        {
            ciclo = false;
        }
    }
    obs.dano = 3;
    obs.cont = 0;
    obs.SizeY = al_get_bitmap_height(bitmap: map);
    obs.SizeX = al_get_bitmap_width(bitmap: map);
    obs.estado = true;
    obs.Siguiente = NULL;
}
```

moverPlayer: Dado la referencia del jugador, los grados de rotación, y las dimensiones de la pantalla, se mueve en las coordenadas del coche, solo cuando no están tocando los bordes.

```
void moverPlayer(coche& player, int grados, int RX, int RY) {
    switch (grados) {
    case 180:
        if (player.y >= 90 + player.velocidad + 1)
            player.y -= player.velocidad;
        break;
    case 135:
        if (player.y >= 90 + player.velocidad + 1)
            player.y -= player.velocidad;
        if (player.x >= 200 + player.velocidad + 1)
            player.x -= player.velocidad;
        break;
    case 90:
        if (player.x >= 200 + player.velocidad + 1)
            player.x -= player.velocidad;
        break;
    case 45:
        if (player.y + player.SizeY <= RY + 30)
            player.y += player.velocidad;
        if (player.x >= 200 + player.velocidad + 1)
            player.x -= player.velocidad;
        break;
    case 0:
        if (player.y + player.SizeY <= RY + 30)
            player.y += player.velocidad;
        break;
    case 360:
        if (player.y + player.SizeY <= RY + 30)
            player.y += player.velocidad;
        break;
    case 315:
        if (player.y + player.SizeY <= RY + 30)
            player.y += player.velocidad;
        if (player.x + player.SizeX <= RX - 130 - player.velocidad - 1)
            player.x += player.velocidad;
        break;
    case 270:
        if (player.x + player.SizeX <= RX - 130 - player.velocidad - 1)
            player.x += player.velocidad;
        break;
    case 225:
        if (player.y >= 90 + player.velocidad + 1)
            player.y -= player.velocidad;
        if (player.x + player.SizeX <= RX - 130 - player.velocidad - 1)
            player.x += player.velocidad;
        break;
    defult:
        cout << "Indefinido player." << grados << endl;
    }

}
```

moverBala: Dado un puntero a una bala y los grados(dirección) se mueven las coordenadas de la bala.

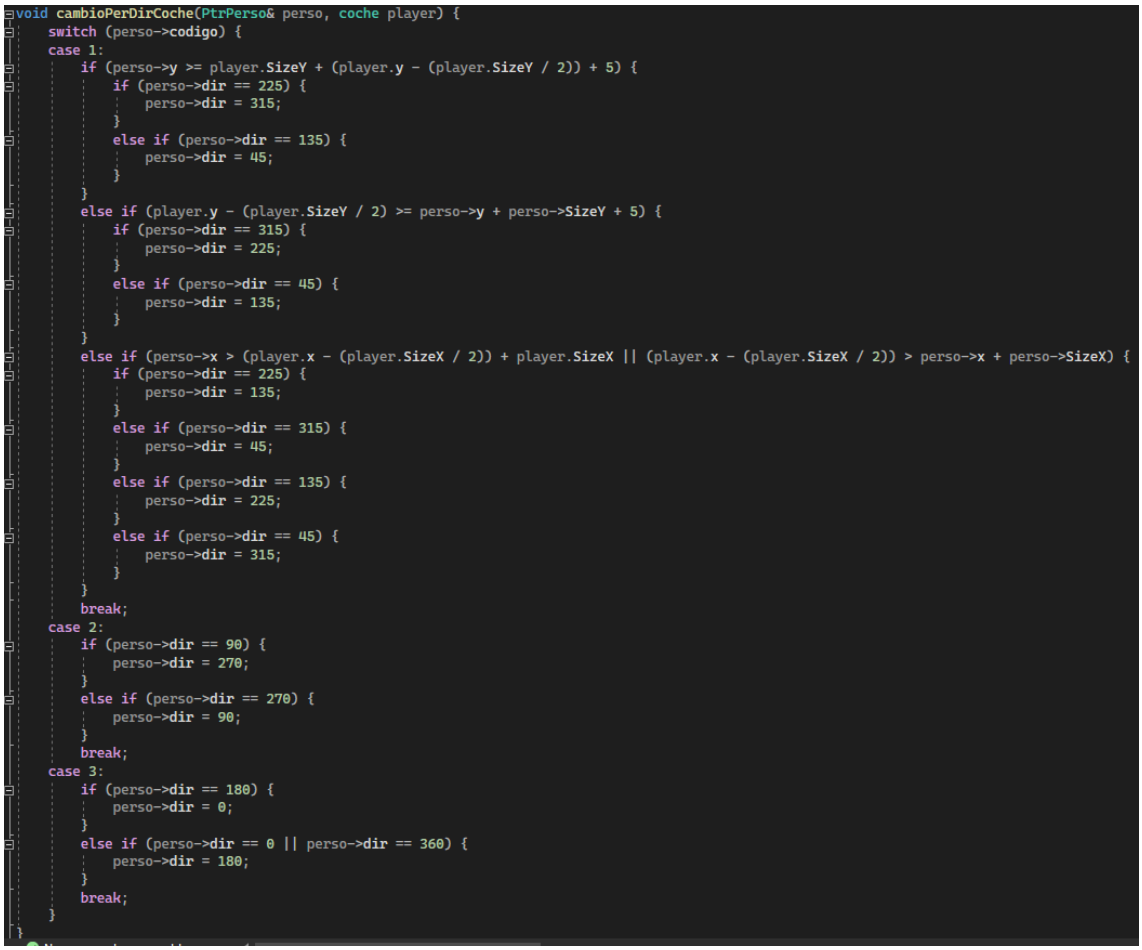


```
void moverBala(PtrBalaEsp bala, int grados) {

    switch (grados) {
    case 180:
        bala->y -= bala->velocidad;
        break;
    case 135:
        bala->y -= bala->velocidad;
        bala->x -= bala->velocidad;
        break;
    case 90:
        bala->x -= bala->velocidad;
        break;
    case 45:
        bala->y += bala->velocidad;
        bala->x -= bala->velocidad;
        break;
    case 0:
        bala->y += bala->velocidad;
        break;
    case 360:
        bala->y += bala->velocidad;
        break;
    case 315:
        bala->y += bala->velocidad;
        bala->x += bala->velocidad;
        break;
    case 270:
        bala->x += bala->velocidad;
        break;
    case 225:
        bala->y -= bala->velocidad;
        bala->x += bala->velocidad;
        break;
    }

}
```

cambioPerDir: Dado una persona y un barril dependiendo de la dirección del movimiento de la persona y su posición con respecto al barril, se busca que rebote de manera consistente.

```
void cambioPerDir(PtrPerso& perso, PtrObsta bars) {
    switch (perso->codigo) {
    case 1:
        if (perso->y >= bars->SizeY + bars->y + 5) {
            if (perso->dir == 225) {
                perso->dir = 315;
            }
            else if (perso->dir == 135) {
                perso->dir = 45;
            }
        }
        else if (bars->y >= perso->y + perso->SizeY + 5) {
            if (perso->dir == 315) {
                perso->dir = 225;
            }
            else if (perso->dir == 45) {
                perso->dir = 135;
            }
        }
        else if (perso->x > bars->x + bars->SizeX || bars->x > perso->x + perso->SizeX) {
            if (perso->dir == 225) {
                perso->dir = 135;
            }
            else if (perso->dir == 315) {
                perso->dir = 45;
            }
            else if (perso->dir == 135) {
                perso->dir = 225;
            }
            else if (perso->dir == 45) {
                perso->dir = 315;
            }
        }
        break;
    case 2:
        if (perso->dir == 90) {
            perso->dir = 270;
        }
        else if (perso->dir == 270) {
            perso->dir = 90;
        }
        break;
    case 3:
        if (perso->dir == 180) {
            perso->dir = 0;
        }
        else if (perso->dir == 0 || perso->dir == 360) {
            perso->dir = 180;
        }
        break;
    }
}
```

cambioPerDirCoche: Se busca el mismo efecto que con la función anterior pero esta vez comparando la persona con el jugador, para que así no se puedan suicidar las personas.



```
void cambioPerDirCoche(PtrPerso& perso, coche player) {
    switch (perso->codigo) {
    case 1:
        if (perso->y >= player.SizeY + (player.y - (player.SizeY / 2)) + 5) {
            if (perso->dir == 225) {
                perso->dir = 315;
            }
            else if (perso->dir == 135) {
                perso->dir = 45;
            }
        }
        else if (player.y - (player.SizeY / 2) >= perso->y + perso->SizeY + 5) {
            if (perso->dir == 315) {
                perso->dir = 225;
            }
            else if (perso->dir == 45) {
                perso->dir = 135;
            }
        }
        else if (perso->x > (player.x - (player.SizeX / 2)) + player.SizeX || (player.x - (player.SizeX / 2)) > perso->x + perso->SizeX) {
            if (perso->dir == 225) {
                perso->dir = 135;
            }
            else if (perso->dir == 315) {
                perso->dir = 45;
            }
            else if (perso->dir == 135) {
                perso->dir = 225;
            }
            else if (perso->dir == 45) {
                perso->dir = 315;
            }
        }
        break;
    case 2:
        if (perso->dir == 90) {
            perso->dir = 270;
        }
        else if (perso->dir == 270) {
            perso->dir = 90;
        }
        break;
    case 3:
        if (perso->dir == 180) {
            perso->dir = 0;
        }
        else if (perso->dir == 0 || perso->dir == 360) {
            perso->dir = 180;
        }
        break;
    }
}
```

moverPintarPer: Esta función recibe una lista de personas, las dimensiones de la pantalla y un bitmap para las personas. Este mueve y pinte a las personas.

```
void moverPintarPer(PtrPerso& perso, int RX, int RY, ALLEGRO_BITMAP* map) {
    PtrPerso Aux;
    Aux = perso;
    while (Aux != NULL)
    {
        if (Aux->estado) {//Si la persona esta viva.
            //Primero verifica si en caso de choca con uno de los bordes del area transitable, cambie la direccion.
            switch (Aux->codigo) {
            case 1:
                if (Aux->x + Aux->SizeX >= RX) {
                    if (Aux->dir == 225) {
                        Aux->dir = 135;
                    }
                    else if (Aux->dir == 315) {
                        Aux->dir = 45;
                    }
                }
                else if (Aux->x <= 0) {
                    if (Aux->dir == 135) {
                        Aux->dir = 225;
                    }
                    else if (Aux->dir == 45) {
                        Aux->dir = 315;
                    }
                }
                else if (Aux->y + Aux->SizeY >= RY) {
                    if (Aux->dir == 45) {
                        Aux->dir = 135;
                    }
                    else if (Aux->dir == 315) {
                        Aux->dir = 225;
                    }
                }
                else if (Aux->y <= 68) {
                    if (Aux->dir == 135) {
                        Aux->dir = 45;
                    }
                    else if (Aux->dir == 225) {
                        Aux->dir = 315;
                    }
                }
                //Dependiendo de la direccion se mueve las coordenadas de la persona.
                switch (Aux->dir) {
                case 135:
                    Aux->y -= Aux->velocidad;
                    Aux->x -= Aux->velocidad;
                    break;
                case 45:
                    Aux->y += Aux->velocidad;
                    Aux->x -= Aux->velocidad;
                    break;
                case 315:
                    Aux->y += Aux->velocidad;
                    Aux->x += Aux->velocidad;
                    break;
                case 225:
                    Aux->y -= Aux->velocidad;
                    Aux->x += Aux->velocidad;
                    break;
                }
                break;
             case 2:
                if (Aux->dir == 90 && Aux->x <= 0) {
                    Aux->dir = 270;
                }
                else if (Aux->dir == 270 && Aux->x + Aux->SizeX >= RX) {
                    Aux->dir = 90;
                }
                switch (Aux->dir) {
                case 90:
                    Aux->x -= Aux->velocidad;
                    break;
                case 270:
                    Aux->x += Aux->velocidad;
                    break;
                }
                break;
```

```
            case 3:
                if ((Aux->dir == 0 || Aux->dir == 360) && Aux->y + Aux->SizeY >= RY)
                    Aux->dir = 180;
                if (Aux->dir == 180 && Aux->y <= 90 + Aux->velocidad + 1) {
                    Aux->dir = 0;
                }
                switch (Aux->dir) {
                case 0:
                    Aux->y += Aux->velocidad;
                    break;
                case 360:
                    Aux->y += Aux->velocidad;
                    break;
                case 180:
                    Aux->y -= Aux->velocidad;
                    break;
                }
                break;

            }
            //Finalmente se imprime la persona en pantalla.
            al_draw_bitmap(bitmap: map, dx: Aux->x, dy: Aux->y, flags: NULL);
        }
        else {//Si la persona esta muerta, si el contador de reaparicion cuenta mayor o igual a 120, revivi la personas y le asigna coordenadas nuevas.
            if (Aux->cont >= 120) {
                Aux->cont = 0;
                Aux->x = rand() % (200 - al_get_bitmap_width(bitmap: map));
                Aux->y = 90 + (rand() % (RY - al_get_bitmap_width(bitmap: map) - 90));
                Aux->estado = true;
            }
        }
        Aux = Aux->Siguiente;
    }
}
```

colisionPlayerObst: Dado un barril y el jugador retorna true, si hay una colisión.

```
bool colisionPlayerObst(coche& player, obsta& barril1) {
    return colisionObjetos(x1: player.x - (player.SizeX / 2), y1: player.y - (player.SizeY / 2), w1: player.SizeX, h1: player.SizeY, x2: barril1.x,
        y2: barril1.y, w2: barril1.SizeX, h2: barril1.SizeY);
}
```

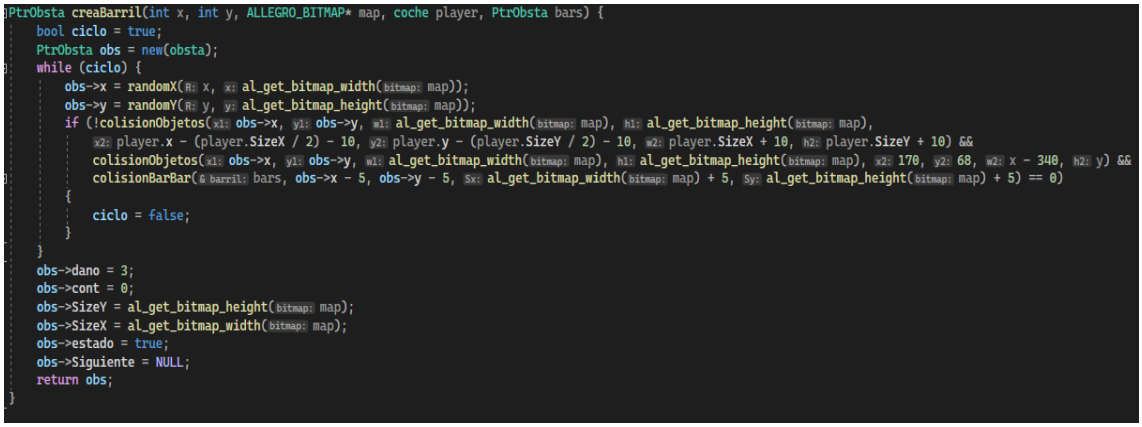
colisionPlayerPersona: Dado una persona y el jugador retorna true, si hay una colisión.

```
bool colisionPlayerPers(coche& player, Persona& perso) {
    return colisionObjetos(x1: player.x - (player.SizeX / 2), y1: player.y - (player.SizeY / 2), w1: player.SizeX, h1: player.SizeY, x2: perso.x,
        y2: perso.y, w2: perso.SizeX, h2: perso.SizeY);
}
```

colisionBarBar: Dado una lista de Barriles y las coordenadas y demisiones de un barril, retorna la cantidad de colisiones que encuentre del barril con la lista de barriles.

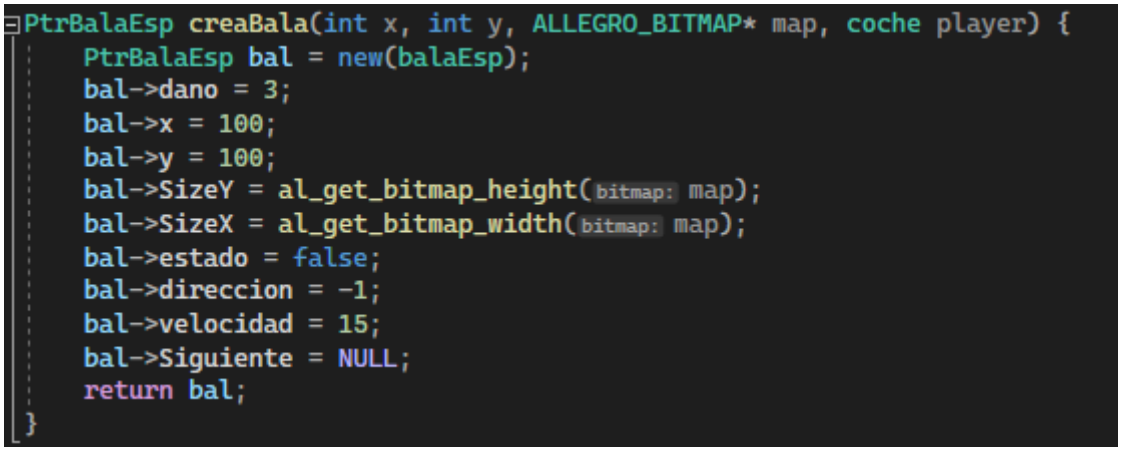
```
int colisionBarBar(PtrObsta& barril, int x, int y, int Sx, int Sy) {
    PtrObsta  Aux;
    Aux = barril;
    int cont = 0;
    while (Aux != NULL)
    {
        if (Aux->estado == true && colisionObjetos(x1: Aux->x, y1: Aux->y, w1: Aux->SizeX, h1: Aux->SizeY, x2: x, y2: y, w2: Sx, h2: Sy)) {
            cont++;
        }
        Aux = Aux->Siguiente;
    }
    return cont;
}
```

crearBarril: Crea un barril con unas coordenadas aleatorias, donde estén en el área jugable, y no colisionen con otro barril o el jugador, y al final retorna el barril.



```
PtrObsta creaBarril(int x, int y, ALLEGRO_BITMAP* map, coche player, PtrObsta bars) {
    bool ciclo = true;
    PtrObsta obs = new(obsta);
    while (ciclo) {
        obs->x = randomX(R: x, x: al_get_bitmap_width(bitmap: map));
        obs->y = randomY(R: y, y: al_get_bitmap_height(bitmap: map));
        if (!colisionObjetos(x1: obs->x, y1: obs->y, w1: al_get_bitmap_width(bitmap: map), h1: al_get_bitmap_height(bitmap: map),
            x2: player.x - (player.SizeX / 2) - 10, y2: player.y - (player.SizeY / 2) - 10, w2: player.SizeX + 10, h2: player.SizeY + 10) &&
            colisionObjetos(x1: obs->x, y1: obs->y, w1: al_get_bitmap_width(bitmap: map), h1: al_get_bitmap_height(bitmap: map), x2: 170, y2: 68, w2: x - 340, h2: y) &&
            colisionBarBar(& barril: bars, obs->x - 5, obs->y - 5, Sx: al_get_bitmap_width(bitmap: map) + 5, Sy: al_get_bitmap_height(bitmap: map) + 5) == 0)
        {
            ciclo = false;
        }
    }
    obs->dano = 3;
    obs->cont = 0;
    obs->SizeY = al_get_bitmap_height(bitmap: map);
    obs->SizeX = al_get_bitmap_width(bitmap: map);
    obs->estado = true;
    obs->Siguiente = NULL;
    return obs;
}
```

crearBala: Crea una bala con valores por defecto y la retorna.



```
PtrBalaEsp creaBala(int x, int y, ALLEGRO_BITMAP* map, coche player) {
    PtrBalaEsp bal = new(balaEsp);
    bal->dano = 3;
    bal->x = 100;
    bal->y = 100;
    bal->SizeY = al_get_bitmap_height(bitmap: map);
    bal->SizeX = al_get_bitmap_width(bitmap: map);
    bal->estado = false;
    bal->direccion = -1;
    bal->velocidad = 15;
    bal->Siguiente = NULL;
    return bal;
}
```

creaPersona: Crea una persona dándoles de manera aleatoria una dirección, y una coordenada random dentro del área transitable.

```
PtrPerso creaPersona(int x, int y, ALLEGRO_BITMAP* map, coche player, int direc) {
    bool ciclo = true;
    PtrPerso per = new(Persona);
    per->cont = 0;
    per->x = rand() % (200 - al_get_bitmap_width(bitmap: map));
    per->y = 90 + (rand() % (y - al_get_bitmap_width(bitmap: map) - 90));
    per->SizeY = al_get_bitmap_height(bitmap: map);
    per->SizeX = al_get_bitmap_width(bitmap: map);
    per->estado = true;
    if (direc == 0) {
        per->codigo = 1;
        switch (rand() % 4) {
        case 0:
            per->dir = 225;
            break;
        case 1:
            per->dir = 45;
            break;
        case 2:
            per->dir = 135;
            break;
        case 3:
            per->dir = 315;
            break;
        }
    }
    per->velocidad = 5;
    per->Siguiente = NULL;
    return per;
}
```

creaMancha: Crea un elemento mancha dados unas coordenadas

```
PtrMancha creaMancha(int x, int y) {
    PtrMancha manch = new(mancha);
    manch->x = x;
    manch->y = y;
    manch->Siguiente = NULL;
    return manch;
}
```

Funciones que añaden elementos de un tipo a una lista correspondiente

```
void anadirBarril(PtrObsta& barril, PtrObsta& nuevo) {
    nuevo->Siguiente = barril;
    barril = nuevo;
}

void anadirBala(PtrBalaEsp& bala, PtrBalaEsp& nuevo) {
    nuevo->Siguiente = bala;
    bala = nuevo;
}

void anadirPersona(PtrPerso& perso, PtrPerso& nuevo) {
    nuevo->Siguiente = perso;
    perso = nuevo;
}

void anadirMancha(PtrMancha& manch, PtrMancha& nuevo) {
    nuevo->Siguiente = manch;
    manch = nuevo;
}
```

imprimirBarril: Imprime los barriles y lleva el conteo del parámetro cont, para la reaparición del barril y la explosión.

```
void imprimirBarril(PtrObsta& barril, ALLEGRO_BITMAP* map, int x, int y, coche player, ALLEGRO_BITMAP* Expl[]) {
    PtrObsta Aux = NULL;
    Aux = barril;
    while (Aux != NULL) {
        if (Aux->estado == true) { //Si el barril esta activo, se imprime en pantalla.
            al_draw_bitmap(bitmap: map, dx: Aux->x, dy: Aux->y, flags: NULL);
        }
        else {
            if (Aux->cont >= 180) { // Si el barril no esta activo y su contador es mayor a 180, se activa,  se le dan coor
                Aux->estado = true;
                bool ciclo = true;
                while (ciclo) {
                    Aux->x = randomX(R: x, x: al_get_bitmap_width(bitmap: map));
                    Aux->y = randomY(R: y, y: al_get_bitmap_height(bitmap: map));
                    if (!colisionObjetos(x1: Aux->x, y1: Aux->y, w1: al_get_bitmap_width(bitmap: map), h1: al_get_bitmap_height
                        ciclo = false;
                    }
                }
                Aux->cont = 0;
            }
            else { // Si esta inactiva y no a pasado del unbral de 180, se reproduce la explosion.
                if (Aux->cont >= 0 && Aux->cont < 5) {
                    al_draw_bitmap(bitmap: Expl[0], dx: Aux->x - ((al_get_bitmap_width(bitmap: Expl[0]) / 2) - (al_get_bitmap_w
                }
                else if (Aux->cont >= 5 && Aux->cont < 10) {
                    al_draw_bitmap(bitmap: Expl[1], dx: Aux->x - ((al_get_bitmap_width(bitmap: Expl[1]) / 2) - (al_get_bitmap_w
                }
                else if (Aux->cont >= 10 && Aux->cont < 15) {
                    al_draw_bitmap(bitmap: Expl[2], dx: Aux->x - ((al_get_bitmap_width(bitmap: Expl[2]) / 2) - (al_get_bitmap_w
                }
                else if (Aux->cont >= 15 && Aux->cont < 20) {
                    al_draw_bitmap(bitmap: Expl[3], dx: Aux->x - ((al_get_bitmap_width(bitmap: Expl[3]) / 2) - (al_get_bitmap_w
                }
                else if (Aux->cont >= 20 && Aux->cont < 25) {
                    al_draw_bitmap(bitmap: Expl[4], dx: Aux->x - ((al_get_bitmap_width(bitmap: Expl[4]) / 2) - (al_get_bitmap_w
                }
                else if (Aux->cont >= 25 && Aux->cont < 30) {
                    al_draw_bitmap(bitmap: Expl[5], dx: Aux->x - ((al_get_bitmap_width(bitmap: Expl[5]) / 2) - (al_get_bitmap_w
                }
                Aux->cont++; //Siempre se aumenta el contador.
            }

        }
        Aux = Aux->Siguiente;
    }

}
```

ColiBar: Se recorre la lista de barriles y si hay alguna colision con el jugador se desactiva el barril, y se le resta a la vida del jugador el daño del barril.

```
void ColiBar(PtrObsta& barril, coche& player, ALLEGRO_SAMPLE* boomBarril) {
    PtrObsta Aux = NULL;
    Aux = barril;
    while (Aux != NULL) {
        if (Aux->estado == true) {
            if (colisionPlayerObst(&: player, & barril1: *Aux)) {

                al_play_sample(data: boomBarril, gain: 0.3, pan: 0, speed: 1, loop: ALLEGRO_PLAYMODE_ONCE, ret_id: NULL);

                Aux->estado = false;
                player.vida -= Aux->dano;

            }
        }
        Aux = Aux->Siguiente;
    }

}
```

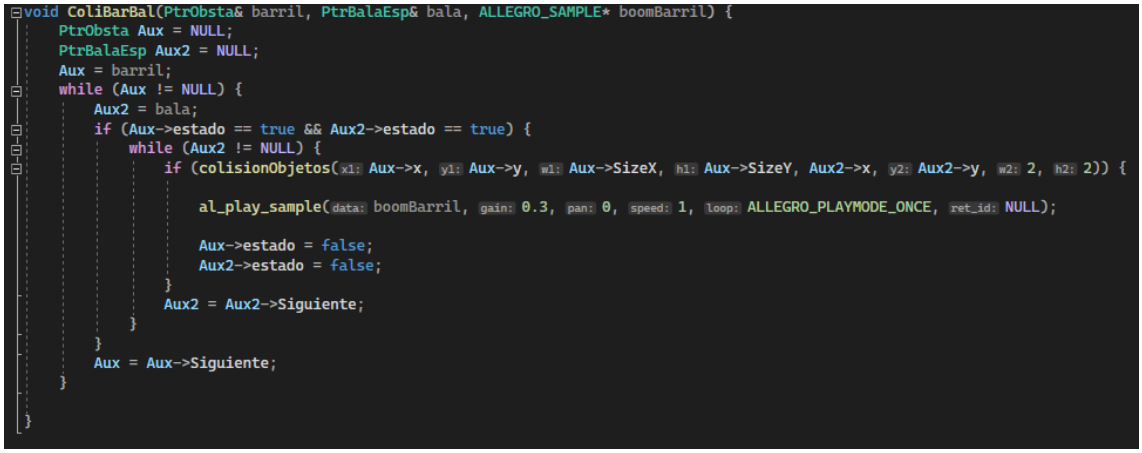
ColiPer: Se recorre la lista de personas y si hay alguna colision con el jugador se desactiva la persona, se suman 10 puntos a los puntos y 1 a los homicidios.

```
void ColiPer(PtrPerso& perso, coche& player, ALLEGRO_SAMPLE* golpePersona) {
    PtrPerso Aux = NULL;
    Aux = perso;
    while (Aux != NULL) {
        if (Aux->estado == true) {
            if (colisionPlayerPers(&: player, & perso: *Aux)) {

                al_play_sample(data: golpePersona, gain: 0.3, pan: 0, speed: 1, loop: ALLEGRO_PLAYMODE_ONCE, ret_id: NULL);

                Aux->estado = false;
                puntos += 10;
                homi++;
            }
        }
        Aux = Aux->Siguiente;
    }
}
```

coliBarPer: Se recorren la lista de personas y barriles, y si hay una colision de personas con barriles o personas con el jugador se cambia la dirección de la persona.

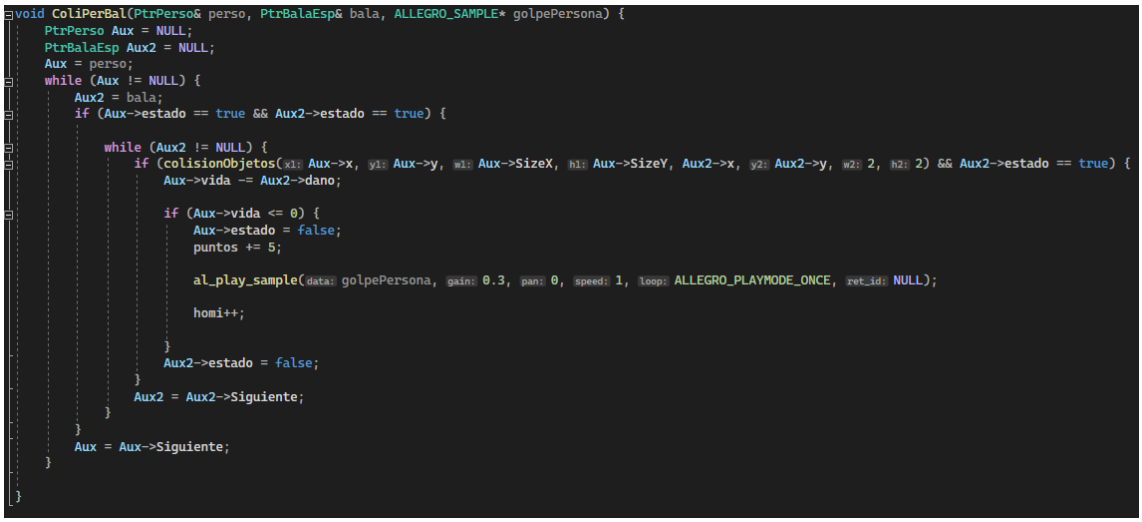
```
void ColiBarPer(PtrObsta& barril, PtrPerso& perso, coche& player) {
    PtrObsta Aux = NULL;
    PtrPerso Aux2 = NULL;
    Aux = barril;
    while (Aux != NULL) {
        Aux2 = perso;
        if (Aux->estado == true) {
            while (Aux2 != NULL) {
                if (colisionObjetos(x1: Aux->x - 15, y1: Aux->y - 15, w1: Aux->SizeX + 15, h1: Aux->SizeY + 15, Aux2->x - 10, y2: Aux2->y - 10, w2: Aux2->SizeX + 10, h2
                    cambioPerDir(& perso: Aux2, bars: Aux);
                }
                if (colisionObjetos(x1: Aux2->x - 25, y1: Aux2->y - 25, w1: Aux2->SizeX + 25, h1: Aux2->SizeY + 25, x2: player.x - (player.SizeX / 2) - 5, y2: player.y
                    cambioPerDirCoche(& perso: Aux2, player);
                }
                Aux2 = Aux2->Siguiente;
            }
        }
        Aux = Aux->Siguiente;
    }
}
```

ColiBarBal: Se recorren balas y barriles y se ambos están activos y hay una colision se desactivan los dos.



```
void ColiBarBal(PtrObsta& barril, PtrBalaEsp& bala, ALLEGRO_SAMPLE* boomBarril) {
    PtrObsta Aux = NULL;
    PtrBalaEsp Aux2 = NULL;
    Aux = barril;
    while (Aux != NULL) {
        Aux2 = bala;
        if (Aux->estado == true && Aux2->estado == true) {
            while (Aux2 != NULL) {
                if (colisionObjetos(x1: Aux->x, y1: Aux->y, w1: Aux->SizeX, h1: Aux->SizeY, Aux2->x, y2: Aux2->y, w2: 2, h2: 2)) {

                    al_play_sample(data: boomBarril, gain: 0.3, pan: 0, speed: 1, loop: ALLEGRO_PLAYMODE_ONCE, ret_id: NULL);

                    Aux->estado = false;
                    Aux2->estado = false;
                }
                Aux2 = Aux2->Siguiente;
            }
        }
        Aux = Aux->Siguiente;
    }

}
```

ColiPerBal: Se recorren balas y personas y se ambos están activos se resta el daño de la bala a la vida de la persona, se desactiva la bala y si la vida de la persona es menor igual que 0, se desactiva la persona.



```
void ColiPerBal(PtrPerso& perso, PtrBalaEsp& bala, ALLEGRO_SAMPLE* golpePersona) {
    PtrPerso Aux = NULL;
    PtrBalaEsp Aux2 = NULL;
    Aux = perso;
    while (Aux != NULL) {
        Aux2 = bala;
        if (Aux->estado == true && Aux2->estado == true) {

            while (Aux2 != NULL) {
                if (colisionObjetos(x1: Aux->x, y1: Aux->y, w1: Aux->SizeX, h1: Aux->SizeY, Aux2->x, y2: Aux2->y, w2: 2, h2: 2) && Aux2->estado == true) {
                    Aux->vida -= Aux2->dano;

                    if (Aux->vida <= 0) {
                        Aux->estado = false;
                        puntos += 5;

                        al_play_sample(data: golpePersona, gain: 0.3, pan: 0, speed: 1, loop: ALLEGRO_PLAYMODE_ONCE, ret_id: NULL);

                        homi++;

                    }
                    Aux2->estado = false;
                }
                Aux2 = Aux2->Siguiente;
            }
        }
        Aux = Aux->Siguiente;
    }

}
```

DestruirBarriles: Destruye una lista enlazada de elementos tipo obsta, tomada de ejemplos antiguos.

```
void DestruirBarriles(PtrObsta& bar)
{
    PtrObsta Aux;
    Aux = bar;
    while (Aux != NULL)
    {
        bar = bar->Siguiente;
        delete(Aux);
        Aux = bar;
    }
}
```

DestruirBalas: Destruye una lista enlazada de elementos tipo balaEsp, tomada de ejemplos antiguos.

```
void DestruirBalas(PtrBalaEsp& bar)
{
    PtrBalaEsp Aux;
    Aux = bar;
    while (Aux != NULL)
    {
        bar = bar->Siguiente;
        delete(Aux);
        Aux = bar;
    }
}
```

DestruirPeros: Destruye una lista enlazada de elementos tipo Persona, tomada de ejemplos antiguos.

```
void DestruirPeros(PtrPerso& bar)
{
    PtrPerso Aux;
    Aux = bar;
    while (Aux != NULL)
    {
        bar = bar->Siguiente;
        delete(Aux);
        Aux = bar;
    }
}
```

DestruirMancha: Destruye una lista enlazada de elementos tipo Mancha, tomada de ejemplos antiguos.

```
void DestruirMancha(PtrMancha& bar)
{
    PtrMancha Aux;
    Aux = bar;
    while (Aux != NULL)
    {
        bar = bar->Siguiente;
        delete(Aux);
        Aux = bar;
    }
}
```

CrearArchivo: Se crea la función CrearArchivo que guarda el nombre, el puntaje y el tiempo en un archivo en memoria secundaria.

```
void CrearArchivo(char* nombre, char* tiempo, char* puntaje)
{
    FILE* archivo;
    archivo = fopen(_FileName: "resultados.txt", _Mode: "a");

    if (NULL == archivo) {
        fprintf(_Stream: stderr, _Format: "No se pudo crear archivo %s.\n", "resultados.txt");
        exit(_Code: -1);
    }
    else {
        fprintf(_Stream: archivo, _Format: "Nombre:%s\n", nombre);
        fprintf(_Stream: archivo, _Format: "Tiempo: %s\n", tiempo);
        fprintf(_Stream: archivo, _Format: "Puntaje: %s\n", puntaje);
        fprintf(_Stream: archivo, _Format: "\n\n");
    }
    fclose(_Stream: archivo);
}
```

DestruirInventario: Destruye una lista enlazada de elementos tipo puntaje.

```
void DestruirInventario(PtrResultados& Lista)
{
    PtrResultados Aux;
    Aux = Lista;
    while (Aux != NULL)
    {
        Lista = Lista->Siguiente;
        delete(Aux);
        Aux = Lista;
    }
}
```

CrearArticulo: Crea un objeto tipo resultado que guarda las estadísticas de una partida.

```
PtrResultados CrearArticulo(char* nombre, char* tiempo, char* puntaje)
{
    //crea el elemento
    PtrResultados Pieza = new(resultados);//se crea una pieza

    strcpy(_Destination: Pieza->Nombre, _Source: nombre);
    strcpy(_Destination: Pieza->Tiempo, _Source: tiempo);
    strcpy(_Destination: Pieza->Puntaje, _Source: puntaje);

    Pieza->Siguiente = NULL;

    return Pieza;
}
```

AgregarInicioInventario: Agregar al inicio para implementar un push.

```
void AgregarInicioInventario(PtrResultados& Lista, PtrResultados& Nuevo)
{
    Nuevo->Siguiente = Lista;
    Lista = Nuevo;
}
```

Push: Push que agrega un elemento a una lista enlazada en su primera posición.

```
void Push(PtrResultados& Lista, PtrResultados Nuevo) {
    AgregarInicioInventario(&: Lista, &: Nuevo);
}
```

Top: Retorna sin desligar, el primer valor en el tope de la pila, el primero de la lista enlazada.

```
PtrResultados Top(PtrResultados& Lista) {
    return Lista;
}
```

Pop: Retorna desligando, el primer valor en el tope de la pila, el primero de la lista enlazada.

```
PtrResultados Pop(PtrResultados& Lista) {
    PtrResultados Aux = Lista;//Apuntar el
    Lista = Lista->Siguiente;//La lista ti
    Aux->Siguiente = NULL;//Olvidese que a
    return Aux;
}
```

CargarArchivo: Esta función carga el archivo de partidas anteriores e imprime en pantalla las últimas tres.

```
void CargarArchivo(ALLEGRO_FONT* fuente2, int RX) //Se carga el archivo en memoria secundaria a pantalla con el puntaje y nombre escritos en el display.
{
    PtrResultados lista = NULL;
    PtrResultados Nuevo;

    //Buffer para el texto.
    char nombre[40];
    char tiempo[60];
    char puntaje[50];

    FILE* archivo;

    fopen_s( _Stream: &archivo, _FileName: "resultados.txt", _Mode: "r"); //Se abre al archivo.

    int j = 0;
    if (archivo != NULL)
    {
        while (!feof( _Stream: archivo)) {
            fscanf( _Stream: archivo, _Format: "Nombre:%s\n", nombre);
            fscanf( _Stream: archivo, _Format: "Tiempo: %s\n", tiempo);
            fscanf( _Stream: archivo, _Format: "Puntaje: %s\n", puntaje);
            fscanf( _Stream: archivo, _Format: "\n\n");

            Nuevo = CrearArticulo(nombre, tiempo, puntaje);//Se crean los valores.
            Push(&: lista, Nuevo);//Se envia la estrcutura y el elemento nuevo a insertar.
        }

        //Se imprimen las partidas en pantalla.
        for (int x = 1; x <= 3; x++)
        {
            al_draw_text( font: fuente2, color: al_map_rgb( r: 0, g: 300, b: 15), (RX / 6 + 5) + j, y: 200 + 5, flags: NULL, text: "Nombre:");//para dar el efecto de sombra
            al_draw_text( font: fuente2, color: al_map_rgb( r: 250, g: 300, b: 15), (RX / 6) + j, y: 200, flags: NULL, text: "Nombre:");

            al_draw_text( font: fuente2, color: al_map_rgb( r: 0, g: 300, b: 15), (RX / 6 + 5) + j, y: 300 + 5, flags: NULL, text: Top(&: lista)->Nombre);//para dar el efecto de sombra
            al_draw_text( font: fuente2, color: al_map_rgb( r: 250, g: 300, b: 15), (RX / 6) + j, y: 300, flags: NULL, text: Top(&: lista)->Nombre);


            al_draw_text( font: fuente2, color: al_map_rgb( r: 0, g: 300, b: 15), (RX / 6 + 5) + j, y: 400 + 5, flags: NULL, text: "Tiempo:");//para dar el efecto de sombra
            al_draw_text( font: fuente2, color: al_map_rgb( r: 250, g: 300, b: 15), (RX / 6) + j, y: 400, flags: NULL, text: "Tiempo:");

            al_draw_text( font: fuente2, color: al_map_rgb( r: 0, g: 300, b: 15), (RX / 6 + 5) + j, y: 500 + 5, flags: NULL, text: Top(&: lista)->Tiempo);//para dar el efecto de sombra
            al_draw_text( font: fuente2, color: al_map_rgb( r: 250, g: 300, b: 15), (RX / 6) + j, y: 500, flags: NULL, text: Top(&: lista)->Tiempo);


            al_draw_text( font: fuente2, color: al_map_rgb( r: 0, g: 300, b: 15), (RX / 6 + 5) + j, y: 600 + 5, flags: NULL, text: "Puntaje:");//para dar el efecto de sombra
            al_draw_text( font: fuente2, color: al_map_rgb( r: 250, g: 300, b: 15), (RX / 6) + j, y: 600, flags: NULL, text: "Puantaje:");

            al_draw_text( font: fuente2, color: al_map_rgb( r: 0, g: 300, b: 15), (RX / 6 + 5) + j, y: 700 + 5, flags: NULL, text: Top(&: lista)->Puntaje);//para dar el efecto de sombra
            al_draw_text( font: fuente2, color: al_map_rgb( r: 250, g: 300, b: 15), (RX / 6) + j, y: 700, flags: NULL, text: Top(&: lista)->Puntaje);

            j += 550;

            Pop(&: lista);//se elimina el top de la lista
        }
        DestruirInventario(&: lista);

        fclose( _Stream: archivo);
    }
}
```

Main

Para esta parte solo se enfocará en las partes más importantes del main, este crearía la pantalla de allegro donde se van a encontrar las opciones a cada característica implementada en el juego, en este caso serian los resultados, instrucciones, y el propio juego, cada uno de estos botones nos va a redirigir a su respectiva pantalla.

Bucle principal: Se genera un elemento evento del allegro. Se asigna que se espere hasta que haya un evento en la cola de eventos, para pasárselo a evento, y continuar en el código. Si se detecta que se presiona una tecla y es ESC se termina el bucle, también termina en el caso de que seleccione la opción de salir, si es un evento del timer se hace respectivamente lo que contenga

```
bool menu = true; //Booleano para el ciclo while de cerrado del progr
while (menu) //Este while es igual que el de la simulacion solo que s
{
    ALLEGRO_EVENT eventos; //Se genera un elemnto tipo evento.
    al_wait_for_event(colaEventos, ret_event: &eventos);//Se le indica

    if (eventos.type == ALLEGRO_EVENT_MOUSE_AXES)//Si se mueve el mou
    {
        //Se registra la posición en x y y del mouse
        mousex = eventos.mouse.x;
        mousey = eventos.mouse.y;

    }

    //MENU PRINCIPAL
    if (eventos.type == ALLEGRO_EVENT_TIMER) //Si hay un timer.
    {
        if (eventos.timer.source == timer1)
        {
```

Se van a registrar los moviemientos del raton para poder usarlo luego a la hora de dar click en pantalla para que las opciones funcionen como botones.

Para el menu principal se cuenta con el siguiente codigo:

```
//MENU PRINCIPAL
if (eventos.type == ALLEGRO_EVENT_TIMER) //Si hay un timer.
{
    if (eventos.timer.source == timer1)
    {

        if (contadorDeFrames++ >= Delay) {//Se declara un contador de frames que logra avanzar la animación del menú principal
            if (frameactual++ >= maxFrame) {//Al alcansa los frames maximos el frame actual vuelve a 0.
                frameactual = 0;
            }
            contadorDeFrames = 0;
        }

        al_clear_to_color(color: al_map_rgb(r: 0, g: 0, b: 0));//Limpia pantalla.

        //Se reproduce la animacion del fondo en base a los frames.
        if (frameactual == 0) {
            al_draw_scaled_bitmap(bitmap: fondo1Menu, sx: 0, sy: 0, sw: 1920, sh: 1080, dx: 0, dy: 0, dw: RX + 1000, dh: RY + 600, flags: 0);
        }
        if (frameactual == 1) {
            al_draw_scaled_bitmap(bitmap: fondo2Menu, sx: 0, sy: 0, sw: 1920, sh: 1080, dx: 0, dy: 0, dw: RX + 1000, dh: RY + 600, flags: 0);
        }
        if (frameactual == 2) {
            al_draw_scaled_bitmap(bitmap: fondo3Menu, sx: 0, sy: 0, sw: 1920, sh: 1080, dx: 0, dy: 0, dw: RX + 1000, dh: RY + 600, flags: 0);
        }
        //Se imprimen los textos del menu.
        al_draw_text(font: fuente1, color: al_map_rgb(r: 250, g: 250, b: 15), X / 2 + 2, Y / 10 + 5, flags: ALLEGRO_ALIGN_CENTRE, text: "Death Race");//Da efecto s
        al_draw_text(font: fuente1, color: al_map_rgb(r: 250, g: 300, b: 15), X / 2, Y / 10, flags: ALLEGRO_ALIGN_CENTRE, text: "Death Race");
        al_draw_text(font: fuente2, color: al_map_rgb(r: 250, g: 300, b: 15), X / 2, (RY * (250.0 / 768.0)), flags: ALLEGRO_ALIGN_CENTRE, text: "JUGAR");
        al_draw_text(font: fuente2, color: al_map_rgb(r: 250, g: 300, b: 15), X / 2, (RY * (390.0 / 768.0)), flags: ALLEGRO_ALIGN_CENTRE, text: "RESULTADOS");
        al_draw_text(font: fuente2, color: al_map_rgb(r: 250, g: 300, b: 15), X / 2, (RY * (460.0 / 768.0)), flags: ALLEGRO_ALIGN_CENTRE, text: "INSTRUCCIONES");
        al_draw_text(font: fuente2, color: al_map_rgb(r: 250, g: 300, b: 15), X / 2, (RY * (530.0 / 768.0)), flags: ALLEGRO_ALIGN_CENTRE, text: "SALIR");
    }
```

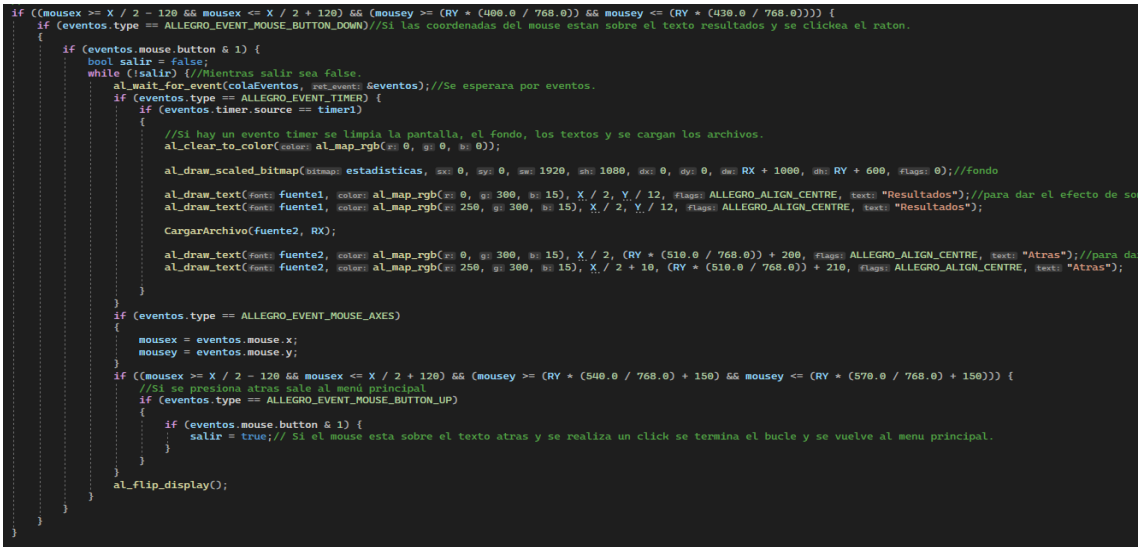
Este se va a estar llamando constantemente si no se detecta ningun evento del raton en las coordenadas establecidas.

Para la opcion Jugar:

```
//BOTON DE JUGAR
if ((mousex >= X / 2 - 42 && mousex <= X / 2 + 42) && (mousey >= (RY * 260.0 / 768.0) && mousey <= (RY * 290.0 / 768.0))) {
    if (eventos.type == ALLEGRO_EVENT_MOUSE_BUTTON_DOWN)//Si las coordenadas del mouse estan sobre el texto jugar y se clickea el raton.
    {
        if (eventos.mouse.button & 1) {

            //Se destruye la pantalla y la musica, y se llama al juego, en caso de acabar el juego, se llama a la funcion main para volver al menu.
            al_destroy_display(display: pantalla);
            al_destroy_sample(spl: musica);
            juego();
            main();//Se vuelve a lanzar el menú si se sale del juego.
            menu = false;
        }
    }
}
```

Cuando el mouse de click al boton de jugar se va a destruir la musica y la pantalla, luego se procede a llamar la funcion juego, esta funcion contaria con una creacion de otra pantalla con allegro y utilizaria las funciones ya explicadas con anteoridad.

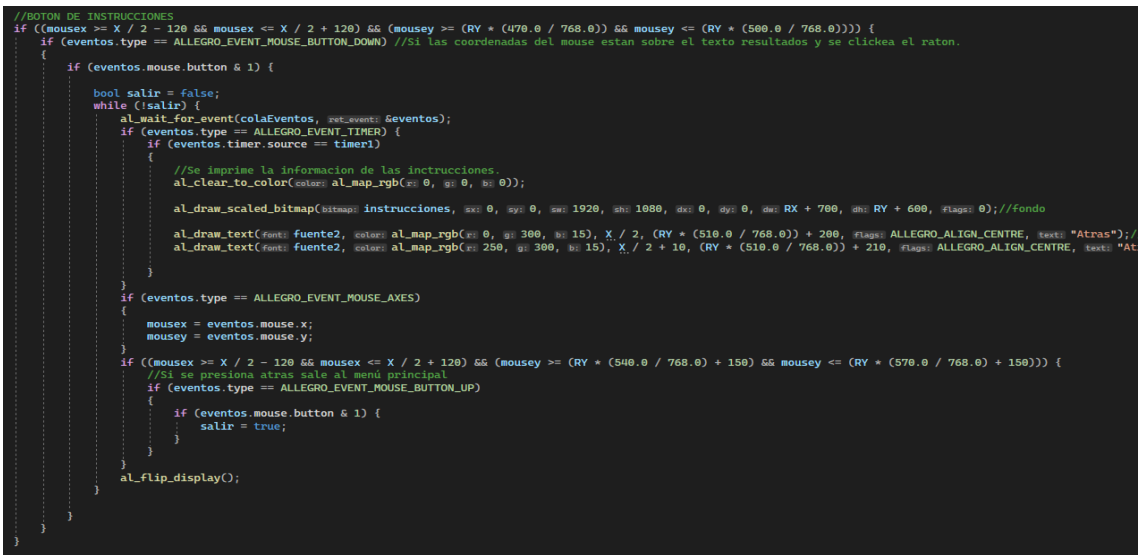
Opción Resultados:



```
if ((mousex >= X / 2 - 120 && mousex <= X / 2 + 120) && (mousey >= (RY * (400.0 / 768.0)) && mousey <= (RY * (430.0 / 768.0)))) {
    if (eventos.type == ALLEGRO_EVENT_MOUSE_BUTTON_DOWN)//Si las coordenadas del mouse estan sobre el texto resultados y se clickea el raton.
    {
        if (eventos.mouse.button & 1) {
            bool salir = false;
            while (!salir) {//Mientras salir sea false.
                al_wait_for_event(colaEventos, ret_event: &eventos);//Se esperara por eventos.
                if (eventos.type == ALLEGRO_EVENT_TIMER) {
                    if (eventos.timer.source == timer1)
                    {
                        //Si hay un evento timer se limpia la pantalla, el fondo, los textos y se cargan los archivos.
                        al_clear_to_color(color: al_map_rgb(r: 0, g: 0, b: 0));

                        al_draw_scaled_bitmap(bitmap: estadisticas, sx: 0, sy: 0, sw: 1920, sh: 1080, dx: 0, dy: 0, dw: RX + 1000, dh: RY + 600, flags: 0);//fondo

                        al_draw_text(font: fuente1, color: al_map_rgb(r: 0, g: 300, b: 15), X / 2, Y / 12, flags: ALLEGRO_ALIGN_CENTRE, text: "Resultados");//para dar el efecto de so
                        al_draw_text(font: fuente1, color: al_map_rgb(r: 250, g: 300, b: 15), X / 2, Y / 12, flags: ALLEGRO_ALIGN_CENTRE, text: "Resultados");

                        CargarArchivo(fuente2, RX);

                        al_draw_text(font: fuente2, color: al_map_rgb(r: 0, g: 300, b: 15), X / 2, (RY * (510.0 / 768.0)) + 200, flags: ALLEGRO_ALIGN_CENTRE, text: "Atras");//para da
                        al_draw_text(font: fuente2, color: al_map_rgb(r: 250, g: 300, b: 15), X / 2 + 10, (RY * (510.0 / 768.0)) + 210, flags: ALLEGRO_ALIGN_CENTRE, text: "Atras");

                    }
                }
                if (eventos.type == ALLEGRO_EVENT_MOUSE_AXES)
                {
                    mousex = eventos.mouse.x;
                    mousey = eventos.mouse.y;
                }
                if ((mousex >= X / 2 - 120 && mousex <= X / 2 + 120) && (mousey >= (RY * (540.0 / 768.0) + 150) && mousey <= (RY * (570.0 / 768.0) + 150))) {
                    //Si se presiona atras sale al menú principal
                    if (eventos.type == ALLEGRO_EVENT_MOUSE_BUTTON_UP)
                    {
                        if (eventos.mouse.button & 1) {
                            salir = true;// Si el mouse esta sobre el texto atras y se realiza un click se termina el bucle y se vuelve al menu principal.
                        }
                    }
                }
                al_flip_display();
            }
        }
    }
}
```

Cuando se posicione el ratón en esta opción y se de click se va a crear otro ciclo donde constantemente se pinta en pantalla los resultados de los 3 últimos jugadores, y si el ratón se posiciona en la palabra Atrás y se da click se procede a volver al menú principal.

Opción Instrucciones:



```
//BOTON DE INSTRUCCIONES
if ((mousex >= X / 2 - 120 && mousex <= X / 2 + 120) && (mousey >= (RY * (470.0 / 768.0)) && mousey <= (RY * (500.0 / 768.0)))) {
    if (eventos.type == ALLEGRO_EVENT_MOUSE_BUTTON_DOWN) //Si las coordenadas del mouse estan sobre el texto resultados y se clickea el raton.
    {
        if (eventos.mouse.button & 1) {

            bool salir = false;
            while (!salir) {
                al_wait_for_event(colaEventos, ret_event: &eventos);
                if (eventos.type == ALLEGRO_EVENT_TIMER) {
                    if (eventos.timer.source == timer1)
                    {
                        //Se imprime la informacion de las inctrucciones.
                        al_clear_to_color(color: al_map_rgb(r: 0, g: 0, b: 0));

                        al_draw_scaled_bitmap(bitmap: instrucciones, sx: 0, sy: 0, sw: 1920, sh: 1080, dx: 0, dy: 0, dw: RX + 700, dh: RY + 600, flags: 0);//fondo

                        al_draw_text(font: fuente2, color: al_map_rgb(r: 0, g: 300, b: 15), X / 2, (RY * (510.0 / 768.0)) + 200, flags: ALLEGRO_ALIGN_CENTRE, text: "Atras");/
                        al_draw_text(font: fuente2, color: al_map_rgb(r: 250, g: 300, b: 15), X / 2 + 10, (RY * (510.0 / 768.0)) + 210, flags: ALLEGRO_ALIGN_CENTRE, text: "At

                    }
                }
                if (eventos.type == ALLEGRO_EVENT_MOUSE_AXES)
                {
                    mousex = eventos.mouse.x;
                    mousey = eventos.mouse.y;
                }
                if ((mousex >= X / 2 - 120 && mousex <= X / 2 + 120) && (mousey >= (RY * (540.0 / 768.0) + 150) && mousey <= (RY * (570.0 / 768.0) + 150))) {
                    //Si se presiona atras sale al menú principal
                    if (eventos.type == ALLEGRO_EVENT_MOUSE_BUTTON_UP)
                    {
                        if (eventos.mouse.button & 1) {
                            salir = true;
                        }
                    }
                }
                al_flip_display();
            }

        }
    }
}
```

Cuando se posicione el ratón en esta opción y se de click se va a crear otro ciclo donde constantemente se pinta en pantalla una imagen con las instrucciones, y si el ratón se posiciona en la palabra Atrás y se da click se procede a volver al menú principal.

Opción Salir:

```
    //BOTON DE SALIR
    if ((mousex >= X / 2 - 120 && mousex <= X / 2 + 120) && (mousey >= (RY * (540.0 / 768.0)) && mousey <= (RY * (570.0 / 768.0)))) {
        //Si se presiona la  opción de salir se sale de la cola de eventos y termina la aplicación.
        if (eventos.type == ALLEGRO_EVENT_MOUSE_BUTTON_DOWN)
        {
            if (eventos.mouse.button & 1) {
                menu = false;
            }
        }
    }

    if (eventos.type == ALLEGRO_EVENT_KEY_DOWN)
    {
        switch (eventos.keyboard.keycode)
        {
        case ALLEGRO_KEY_ESCAPE: //Si se preciona ESCAPE se sale.
            menu = false;
            break;
        }
    }
    else if (eventos.type == ALLEGRO_EVENT_DISPLAY_CLOSE) {
        menu = false; //Si se cierra la ventalla
    }
    al_flip_display();//se muestran los cambios realizados
}

//Se destruyen todos los elementos utilizados en la palicacion.
al_destroy_sample(spl: musica);
al_destroy_timer(timer1);
al_destroy_timer(timer2);
al_destroy_bitmap(bitmap: fondo1Menu);
al_destroy_bitmap(bitmap: fondo2Menu);
al_destroy_bitmap(bitmap: fondo3Menu);
al_destroy_bitmap(bitmap: estadisticas);
al_destroy_bitmap(bitmap: instrucciones);
al_destroy_font(fuente1);
al_destroy_font(fuente2);
al_destroy_event_queue(colaEventos);
```

Si se presiona la tecla Esc se terminaría el código, también si el ratón se posiciona en la opción salir y se da click. Luego se procede a eliminar todas las variables y listas utilizadas.

Juego()

Para esta parte solo se enfocará en las partes más importantes y generales de la función juego(), que representaría la parte jugable del proyecto.

Primero tenemos la configuración general de alegro donde se instalan, inicializan y agregan los componentes básicos para utilizar allegro.

```
if (!al_init()) {
    al_show_native_message_box(NULL, "Ventana Emergente", "Error", "No se puede inicializar Allegro", NULL, NULL);
    return -1;
}
al_init_font_addon();
al_init_ttf_addon();
al_init_image_addon();
al_init_primitives_addon();
al_install_keyboard();
al_install_audio();
al_init_acodec_addon();
al_reserve_samples(7);
ALLEGRO_MONITOR_INFO monitor;
al_get_monitor_info(0, &monitor);
const int RX = monitor.x2 - monitor.x1;
const int RY = monitor.y2 - monitor.y1;

al_set_new_display_flags(ALLEGRO_WINDOWED | ALLEGRO_RESIZABLE | ALLEGRO_FULLSCREEN);
ALLEGRO_DISPLAY* pantalla = al_create_display(RX, RY);

al_set_window_title(pantalla, "Death Race");

if (!pantalla)
{
    al_show_native_message_box(NULL, "Ventana Emergente", "Error", "No se puede crear la pantalla", NULL, ALLEGRO_MESSAGEBOX_ERROR);
    return 0;
}
```

Elementos de allegro (Imágenes, fuentes, sonido, timers)

```
ALLEGRO_FONT* fuente1;
ALLEGRO_FONT* fuente2;
fuente1 = al_load_font("letraDeath.otf", 60, NULL);
fuente2 = al_load_font("letraDeath.otf", 50, NULL);
ALLEGRO_KEYBOARD_STATE teclado;
ALLEGRO_TIMER* timer = al_create_timer(1.0 / FPS);
ALLEGRO_TIMER* timerD = al_create_timer(1);
```

```
ALLEGRO_EVENT_QUEUE* cola_eventos = al_create_event_queue();
ALLEGRO_BITMAP* cochePlayer = al_load_bitmap("Imagenes/Coche.png");
ALLEGRO_BITMAP* persona = al_load_bitmap("Imagenes/Persona2.png");
ALLEGRO_BITMAP* barril = al_load_bitmap("Imagenes/Barril1.png");
ALLEGRO_BITMAP* bullet = al_load_bitmap("Imagenes/Bala.png");
ALLEGRO_BITMAP* fondo2 = al_load_bitmap("Imagenes/Fondo2.png");
ALLEGRO_BITMAP* blood = al_load_bitmap("Imagenes/Blood.png");
ALLEGRO_BITMAP* cora = al_load_bitmap("Imagenes/Corazon.png");
ALLEGRO_BITMAP* Expl[6];
Expl[0] = al_load_bitmap("Imagenes/Ex1.png");
Expl[1] = al_load_bitmap("Imagenes/Ex2.png");
Expl[2] = al_load_bitmap("Imagenes/Ex3.png");
Expl[3] = al_load_bitmap("Imagenes/Ex4.png");
Expl[4] = al_load_bitmap("Imagenes/Ex5.png");
Expl[5] = al_load_bitmap("Imagenes/Ex6.png");
```

```
ALLEGRO_SAMPLE* carro = al_load_sample("Musica/carroMotor.mp3");

ALLEGRO_SAMPLE* explosion = al_load_sample("Musica/explosion.mp3");

ALLEGRO_SAMPLE* disparo = al_load_sample("Musica/disparo.mp3");

ALLEGRO_SAMPLE* boomBarril = al_load_sample("Musica/explosionBarril.mp3");

ALLEGRO_SAMPLE* golpePersona = al_load_sample("Musica/golpePersona.mp3");
```

Fuentes de eventos

```
al_register_event_source(cola_eventos, al_get_timer_event_source(timerD));
al_register_event_source(cola_eventos, al_get_timer_event_source(timer));
al_register_event_source(cola_eventos, al_get_keyboard_event_source());
```

Variables

```
bool hecho = true;
//Para el texto.
char buffer4[5];
char buffer3[5];
char buffer2[5];
char buffer1[10];

//Posicion del jugador
int cochex = 200;
int cochey = 500;

//Numero de balas disparadas.
int contBull = 0;

//Dimensiones del coche.
int cocheLargo = al_get_bitmap_height(cochePlayer);
int cocheAncho = al_get_bitmap_width(cochePlayer);

//Para el movimiento del jugador.
int grados = 180;
int mover = 0;
int desp = 5;

//Tiempo de la partida.
int tiempo = 60;

//Para cuando se termina la partida.
int salida = 3;
```

```
//Las lista enlazadas de objetos
PtrObsta Barriles = NULL;
PtrBalaEsp Balas = NULL;
PtrPerso Personas = NULL;
PtrMancha Manchas = NULL;

PtrMancha nuevo5; // Para generar manchas

//Jugador.
coche player;
```

Inicialización del jugador

```
inicializarPlayer(player, RX, RY, cochePlayer);
```

Creación de los objetos del juego: Personas, obstáculos y balas.

```
for (int i = 0; i < 3; i++) {
    PtrObsta nuevo;
    nuevo = creaBarril(RX, RY, barril, player, Barriles);
    anadirBarril(Barriles, nuevo);
}

for (int i = 0; i < 2; i++) {
    PtrBalaEsp nuevo2;
    nuevo2 = creaBala(RX, RY, barril, player);
    anadirBala(Balas, nuevo2);
}

for (int i = 0; i < 3; i++) {
    PtrPerso nuevo;
    nuevo = creaPersona(RX, RY, persona, player, 0);
    anadirPersona(Personas, nuevo);
}
```

Ciclo principal

Evento presionar tecla: Estos eventos serán: Escape para terminar la partida; flecha derecha para rotar el coche del jugador 45 grados en sentido horario; flecha izquierda para rotar el coche del jugador 45 grados en sentido antihorario; flecha de arriba para indicarle al programa que el estado del coche es en movimiento, y finalmente espacio, donde recorre la lista de balas y se encuentra alguna desactivada la activa y le otorga las coordenadas y dirección actual del jugador.

```
case ALLEGRO_KEY_ESCAPE:
    hecho = false;
    salida = 0;
    break;
case ALLEGRO_KEY_LEFT:
    grados -= 45;
    if (grados <= 0) {
        grados = 360;
    }
    break;
case ALLEGRO_KEY_RIGHT:
    grados += 45;
    if (grados >= 360) {
        grados = 0;
    }
    break;
case ALLEGRO_KEY_UP:
    mover = 1;
    break;
```

```
case ALLEGRO_KEY_SPACE:
    PtrBalaEsp Aux;
    Aux = Balas;
    bool ciclo = true;
    while (Aux != NULL && ciclo)
    {
        if (Aux->estado == false && contBull >= 10) {
            Aux->estado = true;
            Aux->direccion = grados;
            Aux->x = player.x;
            Aux->y = player.y;
            contBull = 0;
            ciclo = true;
            contBull++;
            al_play_sample(disparo, 0.5, 0, 1, ALLEGRO_PLAYMODE_ONCE, NULL);
        }
        else {
            Aux = Aux->Siguiente;
        }
    }
    break;
```

Evento levantar tecla: En este caso solo será la flecha de arriba para cesar el movimiento del jugador.

```
if (eventos.type == ALLEGRO_EVENT_KEY_UP)
{
    switch (eventos.keyboard.keycode) {
    case ALLEGRO_KEY_UP:
        mover = 0;
        break;
    }

}
```

Evento timerD: Este trabaja a un 1 tic por segundo por lo tanto lo utilizaremos para llevar la cuenta regresiva del juego.

```
if (eventos.timer.source == timerD) {
    tiempo--;
}
```

Evento timer: Este es un timer que trabaja a 60 tics por segundo por lo este será el principal que contiene las interacciones del juego.

-Mover player: Si mover es 1 y el jugador esta activo, llama a la función que cambia las coordenadas del jugador.

```
if (mover == 1 && player.estado) {
    moverPlayer(player, grados, RX, RY);
}
```

-Generar fondo: Se limpia la pantalla en negro y se imprime el fondo del tamaño de la pantalla.

```
al_clear_to_color(al_map_rgb(0, 0, 0));
al_draw_scaled_bitmap(fondo2, 0, 0, al_get_bitmap_width(fondo2), al_get_bitmap_height(fondo2), 0, 0, RX, RY, NULL);
```

-Colisiones: Se realiza una primera tanda de búsqueda de colisiones, entre balas y barriles, el jugador y los barriles y el jugador y las personas.

```
ColiBarBal(&Barriles, &Balas, boomBarril);// BALAS VS BARRILES

ColiBar(&Barriles, &player, boomBarril); // BARRILES VS CARRO

ColiPer(&Personas, &player, golpePersona);// PERSONAS VS CARRO
```

-Pintar mover personas y pintar barriles: Se imprimen los barriles disponibles en pantalla, además de evaluar la lista de personas donde se mueve e imprimen en pantalla.

```
imprimirBarril(&Barriles, map:barril, RX, RY, player, Expl);//Se imprimen los barriles.

moverPintarPer(&Personas, RX, RY, map:persona); // Se pueven y pintan a las personas.
```

-Balas: Se recorren las balas y si están activas se imprimen y mueven. En caso de salirse del rango de juego se desactivan.

```
PtrBalaEsp Aux;
Aux = Balas;
while (Aux != NULL)
{
    if (Aux->estado == true) {
        al_draw_bitmap(bitmap:bullet, dx:Aux->x, dy:Aux->y, flags:NULL);
        moverBala(bala:Aux, grados:Aux->direccion);
        if (!colisionObjetos(x1:Aux->x, y1:Aux->y, w1:10, h1:10, x2:170, y2:68, w2:RX - 340, h2:RY)) {
            Aux->estado = false;
        }
    }
    Aux = Aux->Siguiente;
}
```

-Manchas: Se recorren las manchas de sangre y se imprimen en pantalla.

```
PtrMancha Aux3;
Aux3 = Manchas;
while (Aux3 != NULL)
{
    al_draw_bitmap_region(bitmap:blood, sx:150, sy:150, sw:75, sh:75, dx:Aux3->x, dy:Aux3->y, flags:NULL);
    Aux3 = Aux3->Siguiente;
}
```

-Personas: Se recorren las personas y se estas están desactivas se genera la animación de sangre en base al atributo cont. Además de que, al llegar al último fragmento, se genera una mancha para que se mantenga la mancha.

```
PtrPerso Aux2;
Aux2 = Personas;
while (Aux2 != NULL)
{
    if (Aux2->estado == false)
    {
        if (Aux2->cont >= 0 && Aux2->cont < 1) {
            al_draw_bitmap_region(bitmap:blood, sx:0, sy:0, sw:75, sh:75, dx:Aux2->x - (((75 / 2) - (al_get_bitmap_width(bitmap:persona) / 2))), dy:Aux2->y - (((75) / 2) - (al_get_bitmap_height(bitmap:persona) /
        }
        else if (Aux2->cont >= 2 && Aux2->cont < 4) {
            al_draw_bitmap_region(bitmap:blood, sx:75, sy:0, sw:75, sh:75, dx:Aux2->x - (((75 / 2) - (al_get_bitmap_width(bitmap:persona) / 2))), dy:Aux2->y - (((75) / 2) - (al_get_bitmap_height(bitmap:persona) /
        }
        else if (Aux2->cont >= 4 && Aux2->cont < 6) {
            al_draw_bitmap_region(bitmap:blood, sx:150, sy:0, sw:75, sh:75, dx:Aux2->x - (((75 / 2) - (al_get_bitmap_width(bitmap:persona) / 2))), dy:Aux2->y - (((75) / 2) - (al_get_bitmap_height(bitmap:persona)
        }
        else if (Aux2->cont >= 6 && Aux2->cont < 8) {
            al_draw_bitmap_region(bitmap:blood, sx:0, sy:75, sw:75, sh:75, dx:Aux2->x - (((75 / 2) - (al_get_bitmap_width(bitmap:persona) / 2))), dy:Aux2->y - (((75) / 2) - (al_get_bitmap_height(bitmap:persona) /
        }
        else if (Aux2->cont >= 8 && Aux2->cont < 10) {
            al_draw_bitmap_region(bitmap:blood, sx:75, sy:75, sw:75, sh:75, dx:Aux2->x - (((75 / 2) - (al_get_bitmap_width(bitmap:persona) / 2))), dy:Aux2->y - (((75) / 2) - (al_get_bitmap_height(bitmap:persona)
        }
        else if (Aux2->cont >= 10 && Aux2->cont < 12) {
            al_draw_bitmap_region(bitmap:blood, sx:150, sy:75, sw:75, sh:75, dx:Aux2->x - (((75 / 2) - (al_get_bitmap_width(bitmap:persona) / 2))), dy:Aux2->y - (((75) / 2) - (al_get_bitmap_height(bitmap:persona)
        }
        else if (Aux2->cont >= 12 && Aux2->cont < 14) {
            al_draw_bitmap_region(bitmap:blood, sx:0, sy:150, sw:75, sh:75, dx:Aux2->x - (((75 / 2) - (al_get_bitmap_width(bitmap:persona) / 2))), dy:Aux2->y - (((75) / 2) - (al_get_bitmap_height(bitmap:persona)
        }
        else if (Aux2->cont >= 14 && Aux2->cont < 16) {
            al_draw_bitmap_region(bitmap:blood, sx:75, sy:150, sw:75, sh:75, dx:Aux2->x - (((75 / 2) - (al_get_bitmap_width(bitmap:persona) / 2))), dy:Aux2->y - (((75) / 2) - (al_get_bitmap_height(bitmap:persona)
        }
        else if (Aux2->cont == 16) {
            al_draw_bitmap_region(bitmap:blood, sx:150, sy:150, sw:75, sh:75, dx:Aux2->x - (((75 / 2) - (al_get_bitmap_width(bitmap:persona) / 2))), dy:Aux2->y - (((75) / 2) - (al_get_bitmap_height(bitmap:persona
            nuevo5 = creaMancha(Aux2->x - (((75 / 2) - (al_get_bitmap_width(bitmap:persona) / 2))), Aux2->y - (((75) / 2) - (al_get_bitmap_height(bitmap:persona) / 2)));
            anadirMancha(&:Manchas, &:nuevo5);
        }
        Aux2->cont++;
    }
    Aux2 = Aux2->Siguiente;
}
```

-Coli2: Ahora se realiza otra comprobación de colisiones esta ves de las balas con las personas y los barriles además de los barriles con personas.

```
ColiPerBal(&:Personas, &:Balas, golpePersona);// PERSONAS VS BALAS

ColiBarBal(&:Barriles, &:Balas, boomBarril);// BARRILES VS BALAS

ColiBarPer(&:Barriles, &:Personas, &:player);// BARRILES VS PERSONAS
```

-Vida jugador: Se comprueba si la vida del jugador es menor o igual a 0. En caso de serlo se desactiva al jugador.

```
if (player.vida <= 0)
{
    player.estado = false;
}
```

-Imprimir jugador: Se imprime al jugador con la rotación correspondientes si esta activo, sino se va reproducción la animación de explosión y al terminar se termina la partida.

```
if (player.estado) {
    al_draw_rotated_bitmap(bitmap:cochePlayer, cx:player.SizeX / 2, cy:player.SizeY / 2, dx:player.x, dy:player.y, angle:grados * 3.14159 / 180, flags:0);
}
else { //Si esta desactivado en base al contador re reproduce la animacino de explosion y luego se termina la partida.
    if (player.cont >= 0 && player.cont < 5) {
        al_draw_bitmap(bitmap:Expl[0], dx:player.x - ((al_get_bitmap_width(bitmap:Expl[0]) / 2) - (al_get_bitmap_width(bitmap:cochePlayer) / 2)), dy:player.
    }
    else if (player.cont >= 5 && player.cont < 10) {
        al_draw_bitmap(bitmap:Expl[1], dx:player.x - ((al_get_bitmap_width(bitmap:Expl[1]) / 2) - (al_get_bitmap_width(bitmap:cochePlayer) / 2)), dy:player.
    }
    else if (player.cont >= 10 && player.cont < 15) {
        al_draw_bitmap(bitmap:Expl[2], dx:player.x - ((al_get_bitmap_width(bitmap:Expl[2]) / 2) - (al_get_bitmap_width(bitmap:cochePlayer) / 2)), dy:player.
    }
    else if (player.cont >= 15 && player.cont < 20) {
        al_draw_bitmap(bitmap:Expl[3], dx:player.x - ((al_get_bitmap_width(bitmap:Expl[3]) / 2) - (al_get_bitmap_width(bitmap:cochePlayer) / 2)), dy:player.
    }
    else if (player.cont >= 20 && player.cont < 25) {
        al_draw_bitmap(bitmap:Expl[4], dx:player.x - ((al_get_bitmap_width(bitmap:Expl[4]) / 2) - (al_get_bitmap_width(bitmap:cochePlayer) / 2)), dy:player.
    }
    else if (player.cont >= 25 && player.cont < 30) {
        al_draw_bitmap(bitmap:Expl[5], dx:player.x - ((al_get_bitmap_width(bitmap:Expl[5]) / 2) - (al_get_bitmap_width(bitmap:cochePlayer) / 2)), dy:player.

    }
    else {
        salida = 0;

        hecho = false;
    }
    player.cont++;
}
```

-Texto: Se imprime el texto en pantalla.

```
_itoa_s(_Value:tiempo, & _Buffer:buffer3, _Radix:10);//para escribir el tiempo
if (tiempo < 0) {
    al_draw_text(font:fuente1, color:al_map_rgb(r:255, g:300, b:15), RX / 2, y:40, flags:ALLEGRO_ALIGN_CENTRE, text:"0");
    hecho = false;
}
else {
    al_draw_text(font:fuente1, color:al_map_rgb(r:0, g:300, b:15), RX / 2 + 3, y:43, flags:ALLEGRO_ALIGN_CENTRE, text:buffer3);//efecto sombra
    al_draw_text(font:fuente1, color:al_map_rgb(r:255, g:300, b:15), RX / 2, y:40, flags:ALLEGRO_ALIGN_CENTRE, text:buffer3);
}
al_draw_text(font:fuente1, color:al_map_rgb(r:0, g:300, b:15), RX / 2 + 3, y:8, flags:ALLEGRO_ALIGN_CENTRE, text:"Death Race");//efecto sombra
al_draw_text(font:fuente1, color:al_map_rgb(r:255, g:300, b:15), RX / 2, y:5, flags:ALLEGRO_ALIGN_CENTRE, text:"Death Race");

_itoa_s(_Value:puntos, & _Buffer:buffer1, _Radix:10);

al_draw_text(font:fuente2, color:al_map_rgb(r:0, g:300, b:15), RX / 24 + 3, y:13, flags:ALLEGRO_ALIGN_LEFT, text:"Puntuacion:");//efecto sombra
al_draw_text(font:fuente2, color:al_map_rgb(r:255, g:300, b:15), RX / 24 , y:10, flags:ALLEGRO_ALIGN_LEFT, text:"Puntuacion:");

al_draw_text(font:fuente2, color:al_map_rgb(r:0, g:300, b:15), RX / 24 + 3, y:53, flags:ALLEGRO_ALIGN_LEFT, text:buffer1);//efecto sombra
al_draw_text(font:fuente2, color:al_map_rgb(r:255, g:300, b:15), RX / 24, y:50, flags:ALLEGRO_ALIGN_LEFT, text:buffer1);


al_draw_text(font:fuente2, color:al_map_rgb(r:0, g:300, b:15), RX / 6 + 3, y:13, flags:ALLEGRO_ALIGN_LEFT, text:"Homicidio en primer grado:");/
al_draw_text(font:fuente2, color:al_map_rgb(r:255, g:300, b:15), RX / 6, y:10, flags:ALLEGRO_ALIGN_LEFT, text:"Homicidio en primer grado:");

_itoa_s(_Value:homi, & _Buffer:buffer4, _Radix:10);
al_draw_text(font:fuente2, color:al_map_rgb(r:0, g:300, b:15), RX / 4 + 3, y:53, flags:ALLEGRO_ALIGN_LEFT, text:buffer4);//efecto sombra
al_draw_text(font:fuente2, color:al_map_rgb(r:255, g:300, b:15), RX / 4, y:50, flags:ALLEGRO_ALIGN_LEFT, text:buffer4);
```

-Corazones: Se imprimen los corazones en base a la vida del jugador.

```
if (player.vida == 10)
{
    al_draw_bitmap(bitmap:cora, dx:RX - 60, dy:10, flags:NULL);
    al_draw_bitmap(bitmap:cora, dx:RX - 120, dy:10, flags:NULL);
    al_draw_bitmap(bitmap:cora, dx:RX - 180, dy:10, flags:NULL);
    al_draw_bitmap(bitmap:cora, dx:RX - 240, dy:10, flags:NULL);
}
else if (player.vida == 7) {
    al_draw_bitmap(bitmap:cora, dx:RX - 60, dy:10, flags:NULL);
    al_draw_bitmap(bitmap:cora, dx:RX - 120, dy:10, flags:NULL);
    al_draw_bitmap(bitmap:cora, dx:RX - 180, dy:10, flags:NULL);
}
else if (player.vida == 4) {
    al_draw_bitmap(bitmap:cora, dx:RX - 60, dy:10, flags:NULL);
    al_draw_bitmap(bitmap:cora, dx:RX - 120, dy:10, flags:NULL);
}
else if (player.vida == 1) {
    al_draw_bitmap(bitmap:cora, dx:RX - 60, dy:10, flags:NULL);
}
```

Guardar archivos:

Primero se genera una pantalla donde se tomará el nombre del jugador por teclado. Configurando los eventos del mismo:

```
al_clear_to_color(color:al_map_rgb(r:0, g:0, b:0));
if (eventoos.type == ALLEGRO_EVENT_KEY_CHAR) {
    //Para los eventos de teclado si pos, la ubiciacion del caracter es menor a 40, se procede a escribir el nombre
    if (pos != 40) {
        //Si se preciona la tecla BACKSPACE (Borrar) se porra el caracter de la posicion actual y se disminuye en 1 la posicion
        if (eventoos.keyboard.keycode == ALLEGRO_KEY_BACKSPACE) {
            if (pos != 0) {
                nombre[pos] = NULL;
                pos = pos - 1;

            }
        }
```

```
 //Se se preciona ENTER se procede termina el ciclo
 else if (eventoos.keyboard.keycode == ALLEGRO_KEY_ENTER) {
     seguir = false;
 }
 //Se inhabilitan las teclas LEFT, RIGHT, TAB, ESCAPE
 else if (eventoos.keyboard.keycode == ALLEGRO_KEY_LEFT) {
 }
 else if (eventoos.keyboard.keycode == ALLEGRO_KEY_RIGHT) {
 }
 else if (eventoos.keyboard.keycode == ALLEGRO_KEY_TAB) {
 }
 else if (eventoos.keyboard.keycode == ALLEGRO_KEY_ESCAPE) {
 }
 //Si se preciona espacio se procede a imprimer "_"
 else if (eventoos.keyboard.keycode == ALLEGRO_KEY_SPACE) {
     nombre[pos] = char(95);
     pos = pos + 1;
 }
 else {
     //Se no se preciona alguna de las teclas anteriores (se preciona un caracter unicode) se procede a escribirlo en el nombre
     if (eventoos.keyboard.unichar != ALLEGRO_KEY_BACKSPACE) {
         nombre[pos] = eventoos.keyboard.unichar;
         pos = pos + 1;
     }
 }
```

Luego se imprimirá el texto en pantalla.

```
al_draw_text(font:fuente1, color:al_map_rgb(r:255, g:255, b:20), RX / 2, RY / 2 - 250, flags:ALLEGRO_ALIGN_CENTRE, text:"FIN DEL JUEGO");
al_draw_text(font:fuente2, color:al_map_rgb(r:255, g:255, b:255), RX / 2, RY / 2 - 150, flags:ALLEGRO_ALIGN_CENTRE, text:buffer1);
al_draw_text(font:fuente2, color:al_map_rgb(r:255, g:255, b:20), RX / 2, RY / 2, flags:ALLEGRO_ALIGN_CENTRE, text:nombre);
al_draw_text(font:fuente2, color:al_map_rgb(r:255, g:255, b:255), RX / 2, RY / 2 + 200, flags:ALLEGRO_ALIGN_CENTRE, text:"Presione Enter para finalizar");
al_draw_text(font:fuente2, color:al_map_rgb(r:255, g:255, b:255), RX / 2, RY / 2 - 50, flags:ALLEGRO_ALIGN_CENTRE, text:"Ingrese su nombre");
if (nombre[0] == char(95)) {
    al_draw_text(font:fuente2, color:al_map_rgb(r:255, g:255, b:255), RX / 2, RY / 2, flags:ALLEGRO_ALIGN_CENTRE, text:"Escriba su nombre");
}
al_flip_display();
```

Finalmente, al presionar Enter y terminar de ingresar se comprueba el nombre ingresado y se llama a la función CrearArchivo con los datos de la partida convertidos a string.

```
//Se el nombre no es vacio se borra lo que no haya escrito el usuario (como el caracter para ubicar)
if (pos != 0) {
    nombre[pos] = NULL;
}
else {//Si el usuario no escribe el nombre so coloca un signo de interrogacion
    nombre[0] = '?';
}
//Se llama a CrearArchivo con los datos obtenidos
CrearArchivo(nombre, tiempoJ, puntaje);
```

Destrucción:

Finalmente se destruye todos los elementos utilizados en el juego.

```
al_destroy_timer(timer);
al_destroy_timer(timerD);


al_destroy_sample( spl: disparo);
al_destroy_sample( spl: boomBarril);
al_destroy_sample( spl: golpePersona);
al_destroy_sample( spl: explosion);

al_destroy_font(fuente1);
al_destroy_font(fuente2);
DestruirBarriles( &: Barriles);
DestruirBalas( & bar: Balas);
DestruirPeros( & bar: Personas);
DestruirMancha( & bar: Manchas);

al_destroy_display( display: pantalla);

al_destroy_bitmap( bitmap: cochePlayer);
al_destroy_bitmap( bitmap: persona);
al_destroy_bitmap( bitmap: barril);
al_destroy_bitmap( bitmap: bullet);
al_destroy_bitmap( bitmap: fondo2);

al_destroy_event_queue(cola_eventos);
//Se termina el juego.
return salida;
```