UNIVERSIDAD AUTÓNOMA DE SAN LUIS POTOSÍ

VIDEOJUEGO: EL MISTERIO DEL BOSQUE

MANUAL DEL PROGRAMADOR

ALUMNO: MENDOZA RODRÍGUEZ FERNANDO

CLAVE: 285476

MATERIA: ESTRUCTURAS DE DATOS II

SEMESTRE: 2018-2019/I

PROFESOR: DE LA CRUZ LÓPEZ MIGUEL ÁNGEL

1. INTRODUCCIÓN

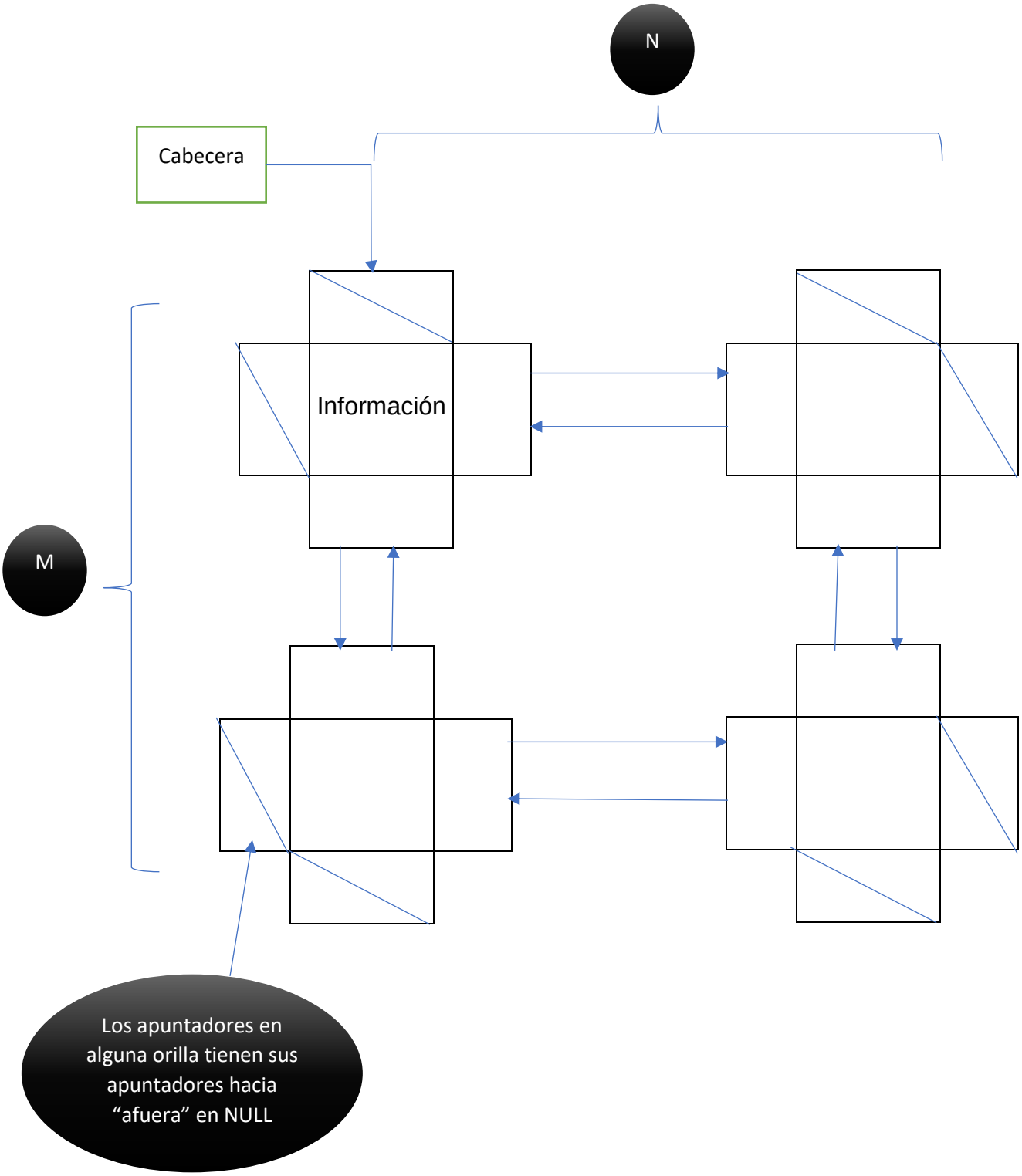
El misterio del bosque es un videojuego hecho en el lenguaje de programación C, con el objetivo de implementar y aprender más acerca de las estructuras de datos, se implementó una lista de listas llamada para conveniencia “malla” entre otras que ya se verán más adelante, habrá ocasiones en las que se mencione el manual del editor de imágenes con el cual se hicieron los sprites, esto ahorrará explicaciones y extenderá el panorama de lo aquí visto

2. DISEÑO

Como se mencionó anteriormente, con el fin de cumplir los objetivos del videojuego se pensaron y diseñaron algunas estructuras de datos.

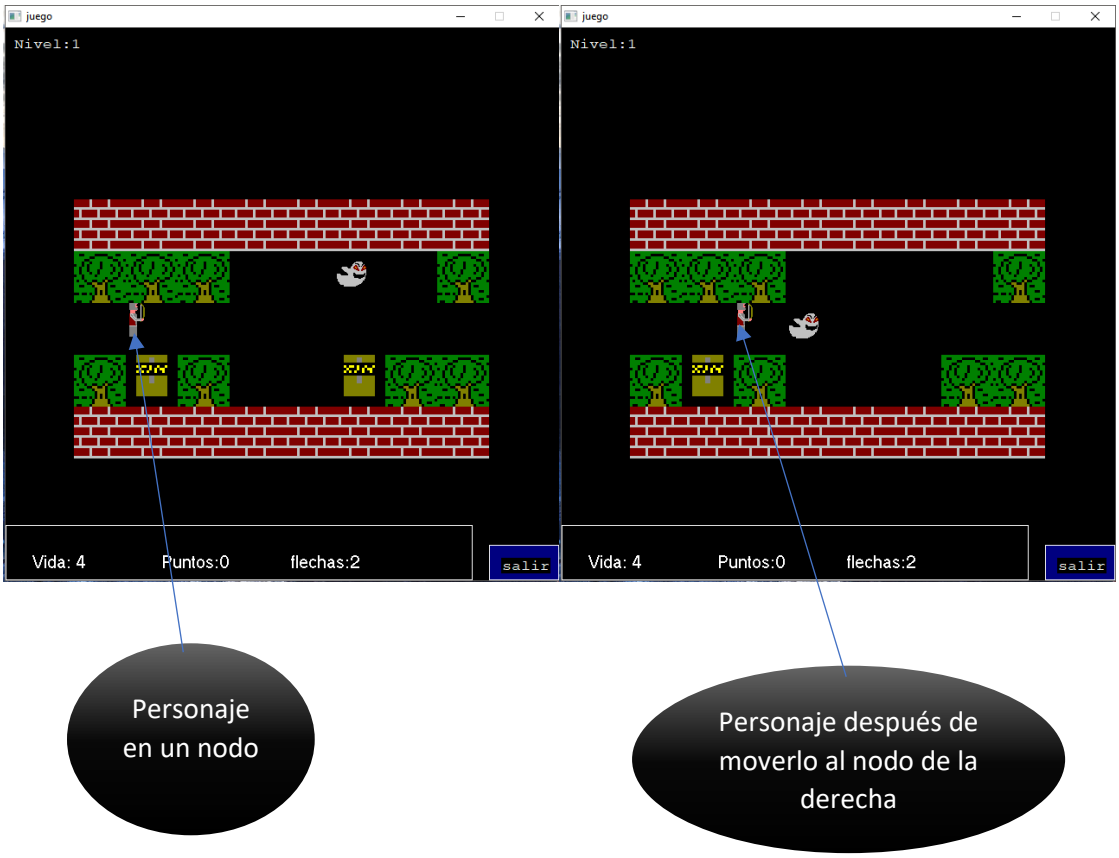
MALLA

Esta es la estructura principal, es la que hace posible el videojuego y facilita todo el trabajo



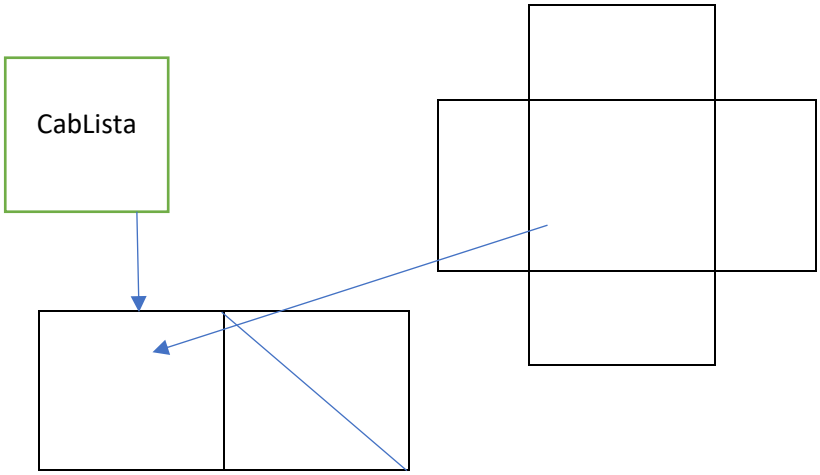
La malla es una especie de lista de listas con los nodos doblemente enlazados entre sí, sirve para poner la información de las coordenadas de comienzo de dibujado de ese nodo, un apuntador al sprite que se dibujara como representación del nodo y una variable para el tipo de sprite al que apunta.

Esta estructura fue implementada porque en el juego se debe mover al personaje y los enemigos de un lugar a otro, por tanto, si se representa un pequeño espacio en la pantalla como un nodo es más fácil, ya que cada nodo tiene sus coordenadas asignadas y está enlazado con las cuatro posiciones a su alrededor, entonces solo se tiene que mover el apuntador y el valor de la variable del tipo de sprite a otro nodo y en la siguiente ocasión se dibujará en dicho nodo.



LISTA

Como su nombre lo dice es una lista en la cual son leídos los archivos de los sprites, en realidad no es necesario leer un sprite en cada nodo de la malla para poder dibujarlo, sería un gasto muy grande de memoria, lo que se prefirió hacer es esta lista, de esa manera cada uno de los nodos que ocupen un sprite solo tendrán que tener un apuntador a el elemento de la lista para poder acceder a él.



BOTON

Esta estructura se hizo porque a lo largo del videojuego se ocupan botones los cuales son más fáciles de implementar si hacemos funciones específicas para ellos, incluyen sus coordenadas de inicio de dibujado (posición en la pantalla), sus dimensiones, color y texto frontal.

RECORD y RECORDS

La estructura RECORD contiene un arreglo para un nombre y una variable para el puntaje, la estructura RECORDS contiene un arreglo de diez elementos del tipo RECORD, esto con el fin de llevar el registro de 10 jugadores que hayan logrado los mejores 10 puntajes.

SPRITE

Esta es la misma estructura que se utilizó en el editor de imágenes. Se implementa para que represente lo que queremos en el juego a nivel visual.

PERSONAJE

Esta estructura contiene un apuntador al nodo en el cual está el personaje, se implementó ya que debido al movimiento del personaje siempre tenemos que tener guardada la posición de la malla en la cual está, así las operaciones del movimiento se realizan más rápido sobre ese nodo.

FLECHA

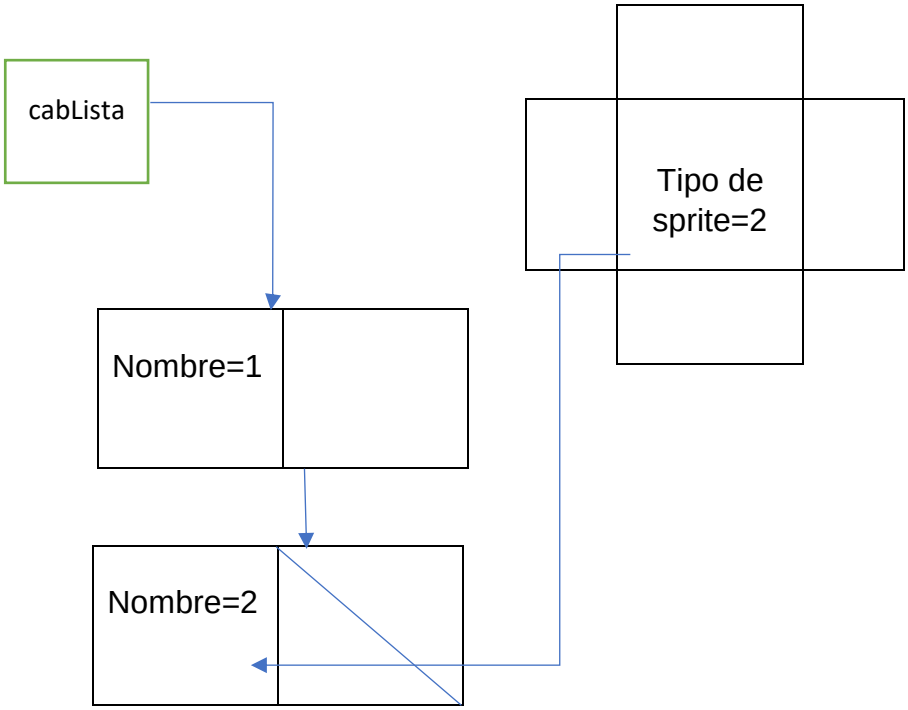
Es otra estructura de tipo lista, pero de flechas, se ha implementado por que es más practico hacer un manejo de las flechas similar al movimiento del personaje por la malla, así se tiene el control absoluto de a dónde y cuándo se mueven las flechas sin tener que buscarlas cada vez que se van a modificar sus posiciones. Se usa porque se necesita trabajar con todas las flechas prácticamente en el mismo instante, es muy fácil lograr esto recorriendo la lista por cada acción.

División del código

El juego está dividido en varios archivos headers con sus respectivos archivos de código fuente

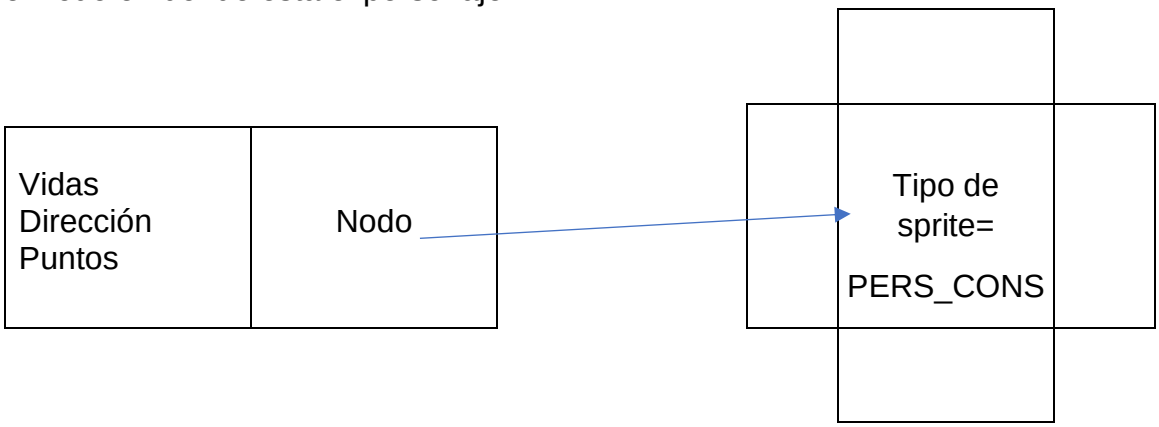
Archivo	Funciones principales
boton.h	- Crear los botones con las especificaciones dadas. - Dibujar los botones. - Checar si un botón fue presionado.
menu.h	- Funcionalidad del menú - Dibujar la pantalla del menú. - Dibujar la pantalla de ayuda. - Dibujar la pantalla de Récords.
records.h	- Leer archivo de récords. - Verificar puntaje de la última partida. - Insertar récord. - Guardar archivo de récords.
text.h	- Obtener una cadena insertada.
personaje.h	- Mover personaje. - Mostrar estadísticas.
sprite.h	- Leer un archivo de sprite. - Dibujar un sprite (al derecho y al reves).
malla.h	- Crear la malla. - Dibujar la malla. - Modificar las coordenadas de la malla. - Eliminar.
juego.h	- Iniciar nivel. - Interacción del juego
flecha.h	- Inicializar lista. - Agregar flecha. - Mover flechas. - Eliminar flechas.

El juego comenzará leyendo los sprites en la lista de sprites, después se lee un nivel de un archivo de texto que contiene los tipos de sprites que van en cada nodo (el archivo de texto se seleccionara en base al nivel que se esté leyendo), mientras se crea la malla el tipo correspondiente es leído del archivo y asignado a cada nodo, inmediatamente se manda una función para buscar el tipo de sprite en la lista y asignar el apuntador en el nodo.



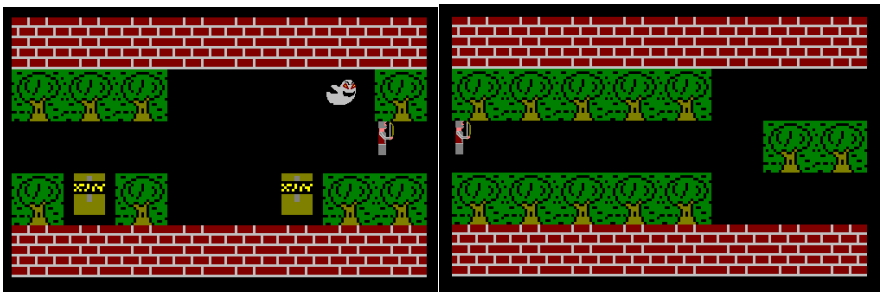
De ese modo teniendo la lista de sprites completa y los documentos de texto para los niveles hechos será muy fácil iniciar un nivel.

Una vez que se inició el nivel se utiliza la estructura PERSONAJE para almacenar el nodo en donde está el personaje.

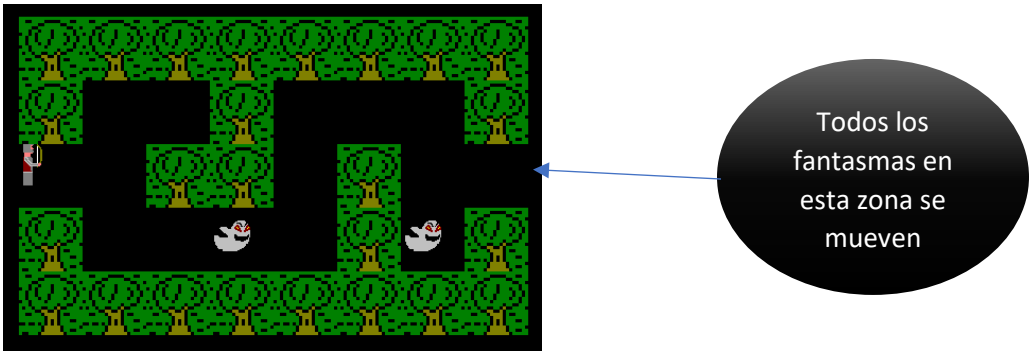


De esa manera se puede copiar rápidamente la información del nodo (El apuntador y el tipo de sprite) a alguno contiguo.

Conforme el personaje se va moviendo a la derecha o izquierda, si aún quedan nodos en esa dirección se hará “scroll”, lo que quiere decir que se cambiaran las coordenadas de dibujado de los nodos para recorrerlos.



Los fantasmas son algo similar, solo que no hay una estructura que guarde sus posiciones exactas, en vez de eso se recorre la malla visible en pantalla buscando esos nodos y se copian a algún nodo contiguo con dirección aleatoria (puede ser arriba, abajo, izquierda o derecha).



Cuando el personaje pase sobre un cofre se obtendrán puntos de manera aleatoria y cuando se presione cierta tecla se lanzará una flecha.

3. IMPLEMENTACIÓN

El videojuego está desarrollado en el lenguaje de programación C con el paradigma estructurado, se usó el IDE (Integrated Development Environment) Code::Blocks en su versión EP, ya que tiene instalada por defecto la biblioteca WinBGIm.

WinBGIm

Esta biblioteca es de vital importancia para la creación del proyecto, en ella están contenidas todas las funciones usadas dentro del proyecto para crear gráficos, como los botones (rectángulos, texto, colores), los sprites (rectángulos, colores), los cuadros de texto etc.

Anteriormente se explicaron las estructuras de datos y las funciones más importantes del juego, ahora se verán en su implementación.

MALLA

La malla como ya fue mencionado es una especie de lista de listas doblemente enlazada.

```
typedef struct nodoMalla
{
    struct nodoMalla*nodoArri;
    struct nodoMalla*nodoAba;
    struct nodoMalla*nodoIzq;
    struct nodoMalla*nodoDer;
    SPRITE *sprt; /***Aquí va la informacion*/
    int tipo;
    int x_I,y_I;//Coordenadas de inicio del segmento de la
malla
}*MALLA;
```

Cada nodo de la malla contiene cuatro apuntadores del tipo del mismo nodo, esto es lo que le da la capacidad de enlazarse doblemente en todas las direcciones.

Se tiene el apuntador a la estructura SPRITE que está dentro de la lista de SPRITES y las coordenadas de comienzo de dibujado para ese nodo.

LISTA

```
typedef struct nodoLista
{
    //El sprite tiene el nombre incluido, por ejemplo si ponemos un apuntador desde un nodo a nube tendremos que buscar
    //a nube para hacer que apunte a este sprite
    SPRITE sprt;
    nodoLista*sig;
}*LISTA;
```

Esta es la estructura de la lista de los sprites, contiene una estructura de tipo sprite y un apuntador a el siguiente elemento.

BOTON

```
typedef struct
{
    //Cordenadas del boton
    int x,y;
    //Color;
    int c;
    //Dimensiones
    int ancho, alto;
    //Texto
    char texto[30];
}BOTON;
```

En esta estructura se tiene todo lo necesario para el uso de un botón, sis características son: coordenadas, color de fondo, dimensiones y el texto.

RECORD y RECORDS

```
typedef struct
{
    char nombre[30];
    int puntaje;
}RECORD;
```

```
typedef struct
{
    RECORD top[TOP];//Arreglo con los 10 mejores
}RECORDS;
```

Una cadena para el nombre y un entero para el puntaje de cada registro.

SPRITE

```
typedef struct
{
    char name[25];
    int row,col;
    int cellsize;
    int pixSize;
    int**grid;
}SPRITE;
```

Esta estructura es heredada del editor de imágenes del cual también se hizo un reporte, se recomienda la lectura de este para ampliar la visión del proyecto, básicamente es una matriz dinámica la cual guarda números que pueden ser interpretados como colores por winBGIm.

PERSONAJE

```
typedef struct
{
    MALLA ptrPer;
    int dir;
    int vida;
    int puntos;
}PERSONAJE;
```

Contiene los datos necesarios para el manejo e interaccion del personaje como lo son su número de vidas, su dirección (izquierda o derecha y sirve para el dibujado) o la cantidad de puntos acumulados.

FLECHA

```
typedef struct nodoFlecha
{
    MALLA nodo;
    struct nodoFlecha*sig;
    int dir;
}*FLECHA;
```

Esta estructura es una lista de flechas, contiene el nodo en el cual será dibujada la flecha, una dirección (Para saber a qué cuadro será movida en el siguiente instante) y una liga hacia la siguiente flecha.

Funciones Principales

boton.h

```
void boton_Inicializate(BOTON*btn,char*texto,int x,int y,int ancho,int alto,int color);
```

- Recibe los parámetros del botón y una estructura botón por referencia.
- Copiará los valores a la estructura.

```
void boton_Dibuja(BOTON bt);
```

- Con las características asignadas dibuja un rectángulo y lo rellena.
- Coloca el texto por encima del botón.

```
int boton_Verificate(BOTON btn);
```

- Esta función verifica que la posición del mouse se encuentre dentro del área ocupada por el botón.

menu.h

```
void menu();
```

- Dentro de ella se crean los botones necesarios para poder imprimirlos y que sirvan como menú, existe un ciclo while en el cual se checan cada uno de los botones cuando se cliquea con el botón izquierdo.
- Se crea la estructura récords y se manda a llamar la función para leerlos

```
while(1)
{
      …
      if(ismouseclick(WM_LBUTTONDOWN))
      {
            …
            If(botón_Verificate(…))
            {
                  …
            }
      }…
}…
```

- Después de presionar un botón se llama a las funciones pertinentes.

```
void dibujaMenu(BOTON bNue,BOTON bRe,BOTON bAyu,BOTON bSal);
```

- Esta función recibe los botones que se usan en el menú principal y manda llamar a la función para dibujarlos, también lee la imagen de fondo y la coloca.

```
void pantallaAyuda();
```

- Se abre un archivo de texto y se imprime carácter por carácter el texto en la pantalla.
- Se siguen ciertas restricciones como que el texto no se salga de un rango de coordenadas.

```
void pantallaRecords(RECORDS recs);
```

- Se recibe la estructura con los récords y se imprimen en pantalla.

text.h

```
void intextxy (int x, int y, char text[]);
```

- Recibe una cadena ciertas coordenadas en donde se verá gráficamente como se introduce el texto.

personaje.h

```
void personaje_Reacomoda(PERSONAJE*per,int dir);
```

- Según el parámetro de la dirección recibido se deside a que nodo se moverá el personaje.

```
void personaje_MuestraEstadisticas(PERSONAJE per,int fle,int nivel);
```

- Recibe los datos actuales del personaje y los imprime en un recuadro debajo de la pantalla de juego.

sprite.h

```
int sprite_Lee(SPRITE*gr,char name[]);
```

- Abre el archivo con el nombre recibido, lee los datos y los copia en el sprite recibido por referencia

```
void sprite_Dibuja(SPRITE gr,int xi, int yi,int pS);
```

```
y void sprite_Dibuja_Reves(SPRITE gr,int xi, int yi,int pS);
```

- Reciben el sprite y lo dibujan en las coordenadas indicadas y con el tamaño de pixel solicitado.

malla.h

```
int malla_Construlle(MALLA*cabM,int nRen,int nCol,LISTA*li,int level);
```

- Recibe la cabecera de la malla por referencia, las dimensiones requeridas y el archivo de texto del cual se le pasará el tipo de sprite al nodo
- Manda llamar a la función de creación de nudo y los enlaza conforme los crea.

```
void malla_Muestra(MALLA cabM)
```

```
y void malla_Remuestra(MALLA cabM,int dir)
```

- Reciben la cabecera de la malla, la primera la dibuja toda la malla a excepción del personaje.
- La segunda dibuja solo los nodos que pueden cambiar de sprite como el personaje, es por eso que se necesita la dirección de este, para saber si se dibuja hacia la izquierda o hacia la derecha.

```
void malla_Reacomoda(MALLA fondo,int dir)
```

- Cambia las coordenadas de todos los nodos dependiendo de la dirección recibida, si es derecha les resta a todos el ancho de pixeles de la pantalla de juego, si es izquierda se los suma.

```
void malla_Elimina(MALLA*cabM);
```

- Elimina de manera recursiva la malla

juego.h

void iniciaNivel(MALLA*f,LISTA*l,PERSONAJE*per,int nivel);

- Recibe la malla vacia por referencia, la lista y el personaje, también el nivel,
- Manda a inicializar y construir la malla, también busca la posición del personaje

void juego(int*puntos)

- Recibe por referencia la variable
- donde se guardaran los puntos de la partida
- Inicia el nivel, dibuja la malla en las dos páginas y pagina solo los nodos de la función remuestra, aquí se controla toda la interacción con el juego.
- Se meve al personaje con estructuras de control y se manda llamar las funciones para configurar los parámetros de las flechas y los fantasmas.

flecha.h

void flecha_Inicializa(FLECHA*cabF);

- Inicializa la lista poniendo NULL.

void flecha_Inserta(FLECHA*cabF,PERSONAJE per);

- Agrega un nuevo elemento a la lista tomando en cuenta la orientación del personaje y su posición en la malla.

void flechas_Modifica(FLECHA cabAuxF);

- Modifica las coordenadas de todas las flechas pasándolas al siguiente nodo

void flechas_Libera (FLECHA*fle,int x_I,int x_F);

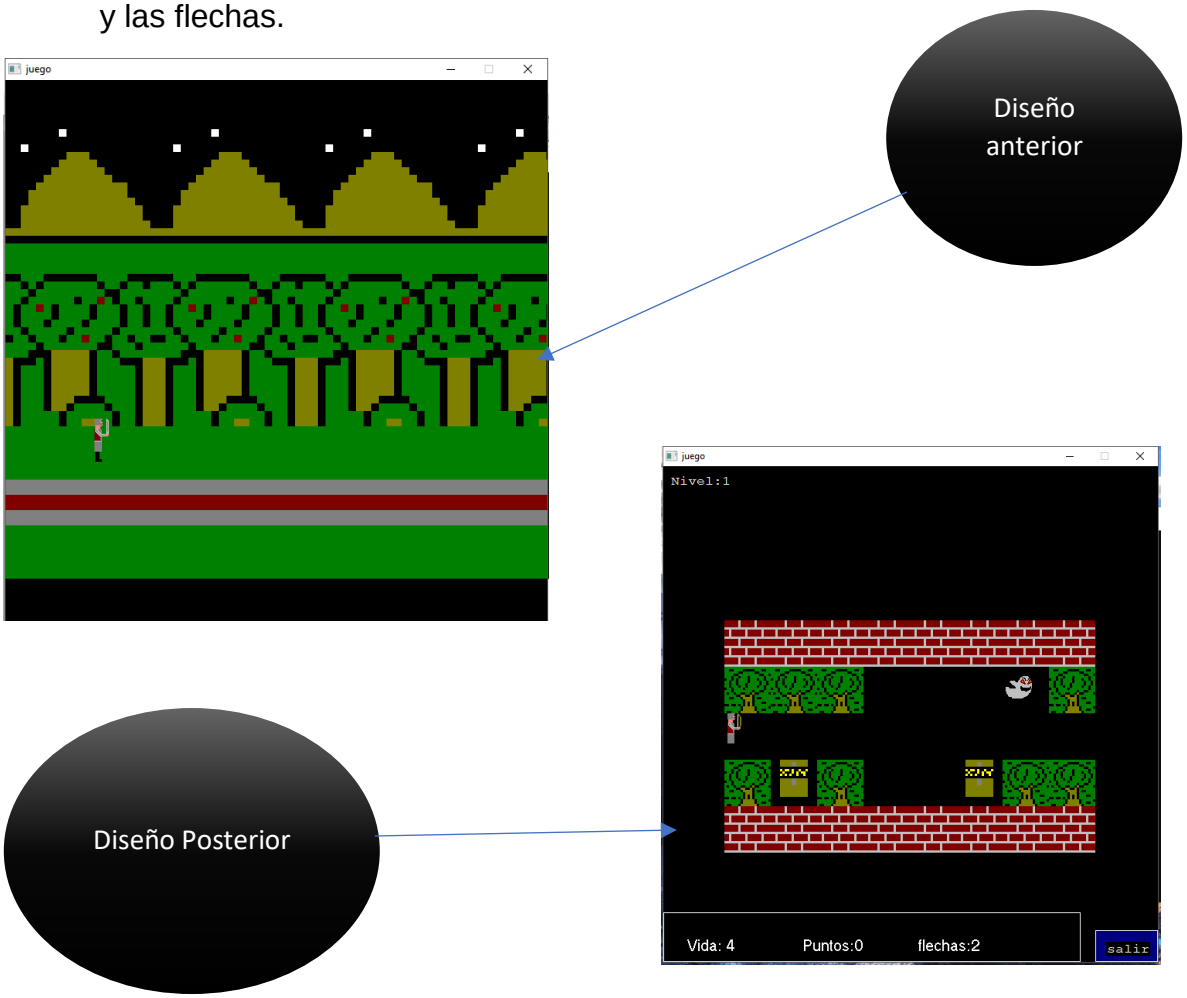
- Quita las flechas que estén fuera del área del juego.

4. LEGADO DEL SISTEMA

Problemas y soluciones

Diseño

- Problema: cuando se estaba diseñando el juego se pensó utilizar la malla solo para el escenario, todo lo demás estaba fuera de la malla, esto era con el fin de dar mejores detalles estéticos y de jugabilidad al juego, sin embargo, el dibujado era tan tardado que las instrucciones de dibujo quedaban almacenadas y se ejecutaban hasta terminar, esto se traduce a que el personaje se seguía moviendo aun cuando el botón no estaba presionado.
- Solución: se implementó el diseño más adecuado que consiste en utilizar los sprites dentro de los nodos, esto incluye también a los personajes enemigos y las flechas.



Rendimiento:

- Problema: Aun cuando el diseño es más simple a nivel gráfico el dibujado era muy lento, por lo cual el personaje se tardaba en moverse.
- Solución: Los nodos se clasifican en dos grupos según el tipo de sprite que contengan, nodos estáticos y nodos con movimiento, los nodos con un árbol o un muro no podrán se accedidos durante el juego, es decir ni el personaje ni los enemigos pueden entrar en ellos, y los nodos restantes pueden ser accedidos de forma normal, de esta manera no se tiene que dibujar el fondo estático cada que se haga el paginado y los demás nodos son los que se paginan.

Restricciones de movimiento:

- Problema: Existen ciertos puntos a los que el personaje no debe moverse, Por ejemplo, un personaje no puede moverse más allá de la malla, ya que no existen nodos a los cuales moverse, tampoco debe moverse al nodo de un árbol o de una barda, esos sprites ya están pensados como estáticos y no se dibujan en el paginado.
- Solución: Se agregaron los condicionales necesarios para que ningún sprite movible pueda realizar ninguna de estas acciones.

Trabajo Futuro

En un futuro sería posible agregarle más armas al juego y más tipos de interacciones como por ejemplo que cuando mata a x enemigos sin que estos lo toquen haya algún ataque especial.

Sería genial que el personaje tuviera un movimiento fluido, en lugar de moverlo de nodo a nodo sin animación habría que aumentar el número de nodos para que pudiera avanzar con alguna especie de animación y al mismo tiempo tener las ventajas de la posición dadas por la malla.

Podría agregarse sonido al juego.

Podría establecerse que al finalizar el nivel los recursos restantes (flechas, vidas) se transformen en puntos.

Calendario de actividades

Actividad	Semana 1 (5 – 11 noviembre)		Semana 2 (12 – 18 noviembre)		Semana 3 (19 – 25 noviembre)		Semana 4 (26 noviembre – 2 diciembre)	
Primer análisis/diseño de lógica de la malla								
Primer diseño visual (sprites)								
Primera implementación de la maya								
Implementación del personaje								
Diseño visual final								
Análisis/diseño final de lógica de la malla								
Implementación de la malla (con personaje y enemigos)								
Implementación del mecanismo de vidas y puntos								
Implementación del mecanismo de movimiento de enemigos								
Implementación de récords								
Implementación de ataque del personaje								

5. REFERENCIAS

University of Colorado, ”Borland Graphics Interface (BGI) for Windows”, http://www.cs.colorado.edu/~main/cs1300/doc/bgi/bgi.html, 2004.