UNIVERSIDAD AUTÓNOMA DE SAN LUIS POTOSÍ

FACULTAD DE INGENIERÍA

“PROYECTO: MEXICA”

ESTRUCTURAS DE DATOS Y ALGORITMOS “B”

PROFR. RAYMUNDO ANTONIO

GONZALEZ GRIMALDO

ALUMNO: EDUARDO MONTES HERNÁNDEZ

ÍNDICE

**Bibliotecas utilizadas………………….……………………………….5**

1. winbgim.h…………………….………………………………5
2. stdio.h……………………….………………………………..5
3. stdlib.h………………………………………………………..5
4. string.h………………………………………………………..5
5. time.h…………………………………………………………5

**Constantes utilizadas……………………………………………...6**

1. REN…………………………………………………………..6
2. COL………………………….……………………………….6
3. TAM\_NODO…………………………………………………6
4. PTLLAX………………………………………………………6
5. PTLLAY………………………………………………………6
6. ESQY………………………………………………………...6
7. ESQX………………………………………………………...6
8. POSX………………………………………………………...6
9. POSY…………………………………………………...........7
10. POSFX……………………………………………….…...7
11. POSFY……………………………………………….…...7
12. INTER…………………………………………...………...7
13. CENT……………………………………………………...7
14. BOTONES………………………………………………...7

**Estructuras utilizadas…………………..……….……………….………8**

1. NODO………………………...……….……………….………8
2. ENEMIGOS………………………….……………….………..8
3. ATAQUE…………………………….………………………....9
4. IMAGEN….……………………….………………………..…..9
5. IMÁGENES…………………………………………………..10
6. PERSONAJE…………………….…………………………..10
7. MENU…………………………………………………...........10
8. RECORDS…………………………………………………...11

**Funciones utilizadas……………………………………………………12**

1. crearMalla…………………………………………………………12
2. insertarInicio………………………………………………………12
3. crearNodo…………………………………………………………12
4. liberarMalla………………………………………………..………12
5. liberarNodo………………………………………………..………13
6. crea\_NodoM………………………………………………..…….13
7. insertBotonM ………………………………………………..……13
8. creaMenu ………………………………………………..……….14
9. dibujaMenu ………………………………………………..……..14
10. liberaMenu …………………………………………………14
11. cargarArchivosImagenes ………………………..……….14
12. abrirImagen ………………………………………………..15
13. crearImagen ………………………………………………..15
14. liberarImagenes ……………………………………………15
15. ordenaBurbuja ……………………………………………..16
16. leerRecords ………………………………………………..16
17. nJugadores …………………………………………………16
18. imprimeRecords ……………………………………………16
19. guardaRecords …………………………………………….17
20. leerArchivoText ………………………………………….…17
21. lee\_texto ………………………………………………..…..17
22. moverPosicionImagen …………………………………….18
23. posicionarJugador …………………………………………18
24. visualizaJuego……………………………………………….19
25. ambienteLV1 ………………………………………………..19
26. dispararAtaqueHeroe ………………………………………20
27. insertarAtaque ………………………………………………20
28. crearAtaque …………………………………………………20
29. moverAtaqueA ………………………………………………21
30. moverAtaqueN ………………………………………………21
31. insertarEnemigo …………………………………………….21
32. crearEnemigo ……………………………………………….22
33. AutomatizarAlien ……………………………………………23
34. reduceVida ………………………………………………….23
35. Nivel ………………………………………………..………..23
36. elimAlien ………………………………………………..……24
37. elimIniAtaque ……………………………………………….24
38. visualizaImagen ……………………………………………24
39. liberaPersonaje …………………………………………….24
40. liberaAtaque ………………………………………………..25
41. limpiaPantalla ………………………………………………25
42. limpiaPag ………………………………………………..….25
43. waitLMC ………………………………………………..……25
44. portada ………………………………………………..…….26
45. leerNombre …………………………………………………26
46. programa ………………………………………………..…..26
47. juego ………………………………………………..……….27
48. perdiste ………………………………………………..…….27

**Función de Sonido ……………………………………………………..28BIBLIOTECAS UTILIZADAS**

1. **<winbgim.h>**

Descripción: Esta librería tiene como objetivo emular la librería graphics.h de Borland C++. En el presente proyecto se utilizaron algunas funciones de esta biblioteca..

1. **<stdio.h>**

Descripción: Es la biblioteca estándar del lenguaje de programación C, el archivo de cabecera que contiene las definiciones de macros, las constantes, las declaraciones de funciones y la definición de tipos usados por varias operaciones estándar de entrada y salida.

1. **<stdlib.h>**

Descripción: Es el archivo de cabecera de la biblioteca estándar de propósito general del lenguaje de programación C. Contiene los prototipos de funciones de C para gestión de memoria dinámica, control de procesos y otras.

1. **<string.h>**

Descripción: Contiene la definición de macros, constantes, funciones y tipos de utilidad para trabajar con cadenas de caracteres y algunas operaciones de manipulación de memoria

1. **<time.h>**

Descripción: Contiene funciones para manipular y formatear la fecha y hora del sistema.

**CONSTANTES UTILIZADAS**

1. **REN**

Descripción: Representa el número de renglones que tendrá la malla durante el juego. Su valor en el proyecto es de 14.

1. **COL**

Descripción: Representa el número de columnas que tendrá la malla durante el juego. Su valor en el proyecto es de 20.

1. **TAM\_NODO**

Descripción: Representa el tamaño de un nodo de la malla. Este sirve para la creación de la malla.

1. **PTLLAX**

Descripción: Es el tamaño que tendrá la pantalla de ancho, este valor se pasa como parámetro a la función initwindow para iniciar la pantalla de gráficos.

1. **PTLLAY**

Descripción: Es el tamaño que tendrá la pantalla de alto, este valor se pasa como parámetro a la función initwindow para iniciar la pantalla de gráficos.

1. **ESQY**

Descripción: Hace referencia al punto de la coordenada en y inicial en donde empezara el margen de la pantalla.

1. **ESQX**

Descripción: Hace referencia al punto de la coordenada en x inicial en donde empezara el margen de la pantalla.

1. **POSY**

Descripción: Es la coordenada en y de donde empezara en menú.

1. **POSX**

Descripción: Es la coordenada en x de donde empezara en menú.

1. **POSFX**

Descripción: Es la coordenada que indica la posición final en x de los botones del menú.

1. **POSFY**

Descripción: Es la coordenada en y que indica la posición final en y del botón del menú.

1. **INTER**

Descripción: Es la altura de cada uno de los botones del menú.

1. **CENT**

Descripción: Es el tamaño que se usa para centrar el texto de los botones del menú.

1. **BOTONES**

Descripción: Indica el número de botones en el menú.

**ESTRUCTURAS UTILIZADAS**

1. **NODO**

Descripción: Esta es la estructura de la malla y representa un nodo de la misma, en ella se almacenan todos los valores que hacen posible determinar ciertas acciones del juego.

|  |  |
| --- | --- |
| VARIABLES | DESCRPCIÓN |
| int xi | Coordenada inicial en x donde comenzará el dibujado de la imagen. |
| int yi | Coordenada inicial en y donde comenzará el dibujado de la imagen. |
| int xf | Coordenada final en x donde finalizará el dibujado de la imagen. |
| int yf | Coordenada final en y donde finalizará el dibujado de la imagen. |
| int estado | Representa si el nodo está ocupado por las distintas imágenes utilizadas en el juego, cada una tiene un valor. |
| int id | Representa si en algún nodo de la malla existe un proyectil. |
| ENEMIGOS \*enemigos | Es un apuntador hacia la lista de enemigos, para facilitar el acceso a la misma |
| struct nodo \*up, \*down, \*left, \*right | Son apuntadores que hacen posible la realización de la malla en un ambiente bidimensional. |

1. **ENEMIGOS**

Descripción: Estructura básica para los enemigos, ésta permite la manipulación de los enemigos en forma de una lista doble.

|  |  |
| --- | --- |
| VARIABLES | DESCRIPCIÓN |
| int índice | Variable que hace posible que el nodo correspondiente obtenga la imagen de un arreglo de imágenes, es decir que haga referencia a la imagen. |
| int dirección | Variable que permite que los enemigos se muevan en distintas direcciones a lo largo y ancho de la malla. |
| int vidas | Cantidad de vidas que posee un enemigo, varia durante el juego y en un momento dado nos avisa que ya llego a cero y entonces se procede a eliminar el nodo de la lista que contiene dicho enemigo. |
| MALLA \*posicion | Apuntador hacia la malla que nos indica en que parte de la misma se encuentra el enemigo. |
| struct \*atrás, \*adelante | Apuntadores hacia la estructura ENEMIGOS que hacen posible la realización de la lista doble de enemigos. |

1. **ATAQUE**

Descripción: estructura básica para los ataques, ésta permite la manipulación de los ataques en forma de lista doble.

|  |  |
| --- | --- |
| VARIABLES | DESCRIPCIÓN |
| int índice | Variable que hace posible que el nodo correspondiente obtenga la imagen de un arreglo de imágenes, es decir que haga referencia a la imagen. |
| int dirección | Variable que permite que los proyectiles se muevan en distintas direcciones a lo largo y ancho de la malla. Valores: del 0 al 3, es decir solo se pueden mover hacia arriba(0), abajo(1), izquierda(2) y derecha(3). |
| MALLA \*posicion | Apuntador hacia la malla que nos indica en que parte de la misma se encuentra el proyectil. |
| struct \*sig \*ant; | Apuntador hacia la estructura ataque que hacen posible la realización de la lista doble de ataques. |

1. **IMAGEN**

Descripción: Estructura básica para guardar una imagen, en forma de arreglo bidimensional de puntos.

|  |  |
| --- | --- |
| VARIABLES | DESCRIPCIÓN |
| int color | Color que se le asignará a cada punto del arreglo. |
| int existe | Variable que nos indica si algún punto del arreglo es visible o invisible, esto nos ayuda para sólo visualizar los puntos que definen la imagen. |

1. **IMÁGENES**

Descripción: Estructura que nos permite la creación de un arreglo de imágenes, es donde se almacenarán todas las imágenes utilizadas en el juego.

|  |  |
| --- | --- |
| VARIABLES | DESCRIPCIÓN |
| IMAGEN \*\*imagen | Variable que hace posible la creación de una imagen a través de un arreglo de bidimensional de puntos. |
| int dimx | Tamaño de la imagen en pixeles a lo ancho. |
| int dimy | Tamaño de la imagen en pixeles a lo alto |

1. **PERSONAJE**

Descripción: Estructura que nos permite la creación y manipulación de cualquiera de los personajes del juego.

|  |  |
| --- | --- |
| VARIABLES | DESCRIPCIÓN |
| int índice | Esta variable nos ayuda en la elección de una imagen para el héroe del juego, accediendo a un arreglo de imágenes a través de este índice. |
| int vidas | Cantidad de vidas que posee el héroe del juego, en un momento dado nos avisa que ya llego a cero y entonces se procede a dar por terminada la partida. |
|  |  |
| ATAQUE \*ataque | Nos permite almacenar en una lista simple todos los ataques que el jugador haya disparado. |
| MALLA \*posición  struct personaje \*ant,\*sig | Apuntador hacia la malla que nos indica en que parte de la misma se encuentra el jugador.  Apuntadores que hacen posible la creación de una lista doble de personajes. |

1. **MENU**

Descripción: Estructura que permite la creación de los botones que se visualizan en el menú principal, estos botones se almacenan en una lista simple.

|  |  |
| --- | --- |
| VARIABLES | DESCRIPCIÓN |
| int num | Indica que número de botón es. Ya que los botones están numerados a partir del 1. Se utiliza para la selección de alguno de los botones. |
| Int x | Indica la posición en x del inicio del botón |
| Int y | Indica la posición en y del inicio del botón |
| Int posx | Indica la posición en x del final del botón |
| Int posy | Indica la posición en y del final del botón |
| Char nom[40] | Almacena el nombre del botón. |
| struct nodoM \*sig | Permite enlazar los nodos y crear una lista simple de botones. |

1. **RECORDS**

Descripción: Estructura básica para los records. Almacena el puntaje del jugador que finalizo el juego y lo coloca en un archivo.

|  |  |
| --- | --- |
| VARIABLES | DESCRIPCIÓN |
| Char nombre[50] | Almacena el nombre del jugador. |
| int puntos | Cantidad de puntos que el jugador logró realizar durante su juego. |

**FUNCIONES UTILIZADAS**

1. **crearMalla**

Prototipo: int crearMalla(MALLA \*\*lM);

Descripción: La tarea fundamental de esta función es la de crear cada uno de los nodos de la malla, por ello es que recibe como parámetro un apuntador por referencia de tipo MALLA, ya que se modificará en el proceso.

|  |  |
| --- | --- |
| PARÁMETROS | DESCRIPCIÓN |
| MALLA \*\*lM | Apuntador por referencia de tipo MALLA, se modificará en la función. |

1. **crearNodo**

Prototipo: MALLA \*crearNodo(int xi, int yi, int xf, int yf);

Descripción: Función que crea un nodo de tipo MALLA. Regresa la dirección del apuntador al nodo creado en caso de que se haya reservado la memoria satisfactoriamente, en caso contrario regresará un valor de NULL;

|  |  |
| --- | --- |
| PARÁMETROS | DESCRIPCIÓN |
| Int xi, yi, xf, yf | Coordenadas iniciales (xi, yi) y finales (xf, yf) que tendrá el nodo en pantalla. |

1. **insertarInicio**

Prototipo: MALLA \*insertarInicio(MALLA \*\*lM, int xi, int yi, int xf, int yf);

Descripción: Lo que hace es añadir o agregar un nodo previamente creado a la malla, con la finalidad de crearla en su totalidad. También regresa la dirección del apuntador al nodo creado, esto con la finalidad de usarlo como auxiliar al momento de enlazar los nodos que se están creando.

|  |  |
| --- | --- |
| PARÁMETROS | DESCRIPCIÓN |
| MALLA \*\*lM | Apuntador por referencia de tipo MALLA, esto con la finalidad de que la cabecera se modifique en el proceso de la función. |
| Int xi, yi, xf, yf | Coordenadas iniciales (xi, yi) y finales (xf, yf) que tendrá el nodo en pantalla. |

1. **liberarMalla**

Prototipo: void liberarMalla(MALLA \*\*malla)

Descripción: La tarea primordial de esta función es liberar el espacio en memoria reservado para la creación de la malla. No regresa ningún valor. Pero recibe el siguiente parámetro por referencia con el objeto de que se logre la liberación de la memoria utilizada previamente.

|  |  |
| --- | --- |
| PARÁMETROS | DESCRIPCIÓN |
| MALLA \*\*malla | Apuntador por referencia a la dirección del primer elemento de la malla. |

1. **liberarNodo**

Prototipo: void liberarNodo(MALLA \*\*malla)

Descripción: Auxiliar en la liberación del espacio en memoria de cada uno de los nodos de la malla. Para ello recibe la dirección del apuntador a un nodo de la malla por referencia con el objeto de que se modifique. No regresa nada excepto la memoria ya liberada.

|  |  |
| --- | --- |
| PARÁMETROS | DESCRIPCIÓN |
| MALLA \*\*malla | Apuntador por referencia a la dirección de algún nodo de la malla. |

1. **creaNodoM**

Prototipo: MENU \*crea\_NodoM(char \*,int x,int y,int xf,int yf,int n);

Descripción: Esta función crea un nodo de tipo MENU y lo regresa con las posiciones en donde se encontrará el botón.

|  |  |
| --- | --- |
| PARÁMETROS | DESCRIPCIÓN |
| Int x, int y, int xf, int yf | Coordenadas iniciales y finales en donde se encontrara el botón del menú |
| int n | Número que se le asignará al botón. |
| char \* | Indica el nombre del botón |

1. **insertaBotonM**

Prototipo: int insertBotonM(MENU \*\*,char \*, int x,int y,int xf,int yf,int n);

Descripción: La función agrega los botones en la lista MENU

|  |  |
| --- | --- |
| PARÁMETROS | DESCRIPCIÓN |
| Int x, int y, int xf, int yf | Coordenadas iniciales y finales en donde se encontrara el botón del menú |
| int n | Número que se le asignará al botón. |
| char \* | Indica el nombre del botón |

1. **creaMenu**

Prototipo: int creaMenu(MENU \*\*M);

Descripción: En esta función se mandan crear todos los botones del programa, recibe una estructura MENU por referencia para poder modificarla con los botones correspondientes del juego.

|  |  |
| --- | --- |
| PARÁMETROS | DESCRIPCIÓN |
| MENU \*\*M | Apuntador por referencia para la creación del menú del juego |

1. **dibujaMenu**

Prototipo: void dibujaMenu(MENU \*M);

Descripción: Se encarga de dibujar el menú principal del juego.

|  |  |
| --- | --- |
| PARÁMETROS | DESCRIPCIÓN |
| MENU \*M | Estructura que contiene los datos necesarios para dibujar el menú principal del juego |

1. **liberaMenu**

Protitipo: int liberaMenu(MENU \*\*M);

Descripción: Al final del juego esta función se encarga de liberar la memoria de los nodos del menú.

|  |  |
| --- | --- |
| PARÁMETROS | DESCRIPCIÓN |
| MENU \*M | Estructura que contiene los datos del menú principal del juego. |

1. **cargarArchivosImagenes**

Prototipo: IMAGENES \*cargarArchivosImagenes(char \*name)

Descripción: Esta función se encarga de leer el archivo donde se encuentra el nombre de las imágenes del juego y al mismo tiempo agregar los datos de éstas en un arreglo y regresarlo.

|  |  |
| --- | --- |
| PARÁMETROS | DESCRIPCIÓN |
| char \*name | Cadena que indica el nombre del archivo donde se encuentran las imagenes |

1. **abrirImagen**

Prototipo: int abrirImagen(IMAGENES \*imagenes, char \*nombre)

Descripción: Esta función recibe el nombre especifico de una imagen para después abrir el archivo con ese nombre y colocar los datos de éste dentro del arreglo de imágenes que recibe.

|  |  |
| --- | --- |
| PARÁMETROS | DESCRIPCIÓN |
| char \*nombre | Cadena que indica el nombre del archivo de la imagen que será agregada al arreglo |
| IMAGENES \*imagenes | Apuntador en donde se agregaran los datos de la imagen |

1. **crearImagen**

Prototipo: IMAGEN \*\*crearImagen(int dimx, int dimy);

Descripción: Reserva espacio para un arreglo bidimensional de puntos de tamaño dimx por tamaño dimy. Retorna el arreglo creado si lo hizo correctamente, de lo contrario retorna NULL;

|  |  |
| --- | --- |
| PARÁMETROS | DESCRIPCIÓN |
| int dimx | Tamaño del arreglo bidimensional en una de sus dimensiones, es decir, en x. |
| int dimy | Tamaño del arreglo bidimensional en una de sus dimensiones, es decir, en y. |

1. **liberarImagenes**

Prototipo: void liberarImagenes(IMAGENES \*\*imágenes);

Descripción: Permite liberar el espacio utilizado en memoria por cada una de las imágenes contenidas en el arreglo. No tiene valor de retorno.

|  |  |
| --- | --- |
| PARÁMETROS | DESCRIPCIÓN |
| IMAGENES \*\*imagenes | Arreglo de imágenes pasado por referencia con el objetivo de liberar el espacio utilizado en memoria. |

1. **ordenaBurbuja**

Prototipo: void ordenaBurbuja( RECORDS \*a , int nJ );

Descripción: Esta función es para ordenar el puntaje de los jugadores, se opera en el momento de imprimir los records recibe como parámetros la estructura y el número de jugadores.

|  |  |
| --- | --- |
| PARÁMETROS | DESCRIPCIÓN |
| RECORDS \*a | Arreglo de RECORDS que se leyeron del archivo de los records del juego |
| Int nJ | Indica el numero de records que hay en el arreglo de records |

1. **leerRecords**

Prototipo: void leerRecords();

Descripción: Lo que hace es leer el archivo de los records para después llamar a la función que los imprime en pantalla.

1. **nJugadores**

Prototipo: int nJugadores();

Descripción: Abre el archivo de los records y lo recorre para regresar el numero de records que hay en el archivo.

1. **imprimeRecords**

Prototipo: void imprimeRecords( RECORDS \*a, int nJ );

Descripción: Por medio de el llamado a esta función se logra imprimir los records del archivo en la pantalla del menú.

|  |  |
| --- | --- |
| PARÁMETROS | DESCRIPCIÓN |
| RECORDS \*a | Arreglo de RECORDS que se leyeron del archivo de los records del juego |
| Int nJ | Indica el numero de records que hay en el arreglo de records |

1. **guardaRecords**

Prototipo: void guardaRecords( RECORDS \*a );

Descripción: Se encarga de recibir una estructura con los records que se agregarán al archivo

|  |  |
| --- | --- |
| PARÁMETROS | DESCRIPCIÓN |
| RECORDS \*a | Estructura que contiene los datos del jugador que se agregaran al archivo de records |

1. **leerArchivoText**

Prototipo: void leerArchivoText( char \*cad, int x, int y);

Descripción: Esta función lee cuaquier archive de texto que reciba como cadena y lo imprime en pantalla dependiendo de las variables “x” y “y” que reciba.

.

|  |  |
| --- | --- |
| PARÁMETROS | DESCRIPCIÓN |
| Char \*cad | Nombre del archivo que se leera |
| Int x | Posición de la coordenada en “x” en donde se empezara a imprimir el archivo |
| Int y | Posición de la coordenada en “x” en donde se empezara a imprimir el archivo |

1. **lee\_texto**

Prototipo: void lee\_texto(int x,int y, char \*cad);

Descripción: Esta function imprime el texto que se va ingresando en la consola de graficos hasta que se presiona la tecla enter y la función guarda la cadena en el parámetro char \*cad que llega por referencia.

|  |  |
| --- | --- |
| PARÁMETROS | DESCRIPCIÓN |
| Char \*cad | Cadena por referencia en la que se guardará el texto que se ingrese. |
| Int x | Posición de la coordenada en “x” en donde se empezara a leer el texto |
| Int y | Posición de la coordenada en “x” en donde se empezara a leer el texto. |

1. **moverPosicionImagen**

Prototipo: void moverPosicionImagen(MALLA \*\*p, int dir, int id);

Descripción: En esta función se lleva a cabo el movimiento del personaje que llegue, cuando se activa se le manda el personaje y la direccion de hacia que parte de la lista se moverá y su id en la malla.

|  |  |
| --- | --- |
| PARÁMETROS | DESCRIPCIÓN |
| int dir | Indica el lado hacia donde se moverá el personaje en la malla |
| int id | Es el id del personaje que se actualizara en la malla |
|  |  |
| MALLA \*\*p | Personaje que se moverá |

1. **posicionarJugador**

Prototipo: void posicionarJugador(PERSONAJE \*\* p, MALLA \*malla, int x, int y, int id);

Descripción: Se hace al inicio del juego para posicionar cada uno de los personajes en el juego, recibe la posición en “x” y “y” del lugan en la malla en donde se le dara la posición al igual que si id que ocupara en la malla.

|  |  |
| --- | --- |
| PARÁMETROS | DESCRIPCIÓN |
| PERSONAJE \*\* p | Personaje que se moverá |
| MALLA \*malla | Indica el apuntador inicial de la malla |
| int x | Campos que se moverá hacia la derecha de la malla |
| int y | Campos que se moverá hacia debajo de la malla |
| int id | Id que ocupara el personaje en la malla |

1. **visualizaJuego**

Prototipo: void visualizaJuego(IMAGENES \*img, PERSONAJE \*heroe, PERSONAJE \*alien, ATAQUE \*ath, PERSONAJE \*naves, PERSONAJE \*lanzas);

Descripción:

|  |  |
| --- | --- |
| PARÁMETROS | DESCRIPCIÓN |
| IMAGENES \*img | Arreglo de las imágenes de juego |
| PERSONAJE \*heroe | Lista que contiene al héroe |
| PERSONAJE \*alien | Lista que contiene a los aliens |
| PERSONAJE \*nave | Lista que contiene las naves |
| PERSONAJE \*heroe | Lista que contiene el ítem de las flechas |
| ATAQUE \*ath | Lista que contiene los ataques del heroe |

1. **ambienteLV1**

Prototipo: void ambienteLV1(RECORDS \*rec, int lanzas, int vidas);

Descripción: Hace la visualización de los datos actuales del juego, las lanzas que se tienen, las vidas, el tiempo, el nombre del jugador y el puntaje actual.

|  |  |
| --- | --- |
| PARÁMETROS | DESCRIPCIÓN |
| RECORDS \*rec | Contiene los datos actuales del jugador, nombre y puntaje |
| int lanzas | Numero que indica las lanzas actuales con las que cuenta el jugador |
| int vidas | Numero que indica las vidas actuales del jugador |

1. **dispararAtaqueHeroe**

Prototipo: int dispararAtaqueHeroe(PERSONAJE \*\*p, int direccion, int tipo);

Descripción: Aquí se lleva a cabo la creación de un ataque para cualquiera de los personajes del juego, recibe la dirección a la que se envía el ataque y el tipo que hace referencia al índice de la imagen que se mostrara del ataque.

|  |  |
| --- | --- |
| PARÁMETROS | DESCRIPCIÓN |
| PERSONAJE \*\*p | Lista a la que se le agregara un ataque. |
| int direccion | Campo que indica mediante los índices 0-Arriba 1-Abajo 2-Izquierda 3-Derecha hacia donde va dirigido el ataque |
| int tipo | Indica el índice de la imagen del ataque |

1. **insertarAtaque**

Prototipo: int insertarAtaque(MALLA \*posicion, ATAQUE \*\*A, int indice, int direccion);

Descripción: Inserta un elemento nuevo en la lista de ataque con su dirección y el índice de la imagen que le corresponde

|  |  |
| --- | --- |
| PARÁMETROS | DESCRIPCIÓN |
| MALLA \*posicion | Posición de la malla en donde se insertará el ataque |
| ATAQUE \*\*A | Lista del personaje a la que se le agregara un ataque |
| int indice | Índice de la imagen del ataque |
| int direccion | Indica la dirección del ataque |

1. **crearAtaque**

Prototipo: ATAQUE \*crearAtaque(MALLA \*posicion, int direccion, int indice);

Descripción: Crea una estructura de tipo ATAQUE a la que se le agregan los campos de la posición de la malla al igual que la dirección y el índice del ataque

|  |  |
| --- | --- |
| PARÁMETROS | DESCRIPCIÓN |
| MALLA \*posicion | Posición en donde se insertara el ataque |
| int direccion | Indica la dirección en donde será lanzado el ataque |
| int indice | Índice de la imagen del ataque |

1. **moverAtaqueA**

Prototipo: void moverAtaqueA(PERSONAJE \*a, PERSONAJE \*\*b, int \*puntos);

Descripción: En esta función se lleva a cabo el movimiento de los ataques tanto enemigos como del héroe en las direcciones izquierda y derecha además de la actualización de los puntos y las vidas del héroe en caso de modificarse.

|  |  |
| --- | --- |
| PARÁMETROS | DESCRIPCIÓN |
| PERSONAJE \*a | Personaje al que se le actualizara el movimiento del ataque |
| PERSONAJE \*\*b | Personaje al que se verificara si le afecta el ataque |
| int \*puntos | Puntaje que será actualizado en caso de ser necesario |

1. **moverAtaqueN**

Prototipo: void moverAtaqueN(PERSONAJE \*a, PERSONAJE \*\*b, int \*puntos);

Descripción: En esta función se lleva a cabo el movimiento de los ataques tanto enemigos como del héroe en las direcciones arriba y abajo además de la actualización de los puntos y las vidas del héroe en caso de modificarse.

|  |  |
| --- | --- |
| PARÁMETROS | DESCRIPCIÓN |
| PERSONAJE \*a | Personaje al que se le actualizara el movimiento del ataque |
| PERSONAJE \*\*b | Personaje al que se verificara si le afecta el ataque |
| int \*puntos | Puntaje que será actualizado en caso de ser necesario |

1. **insertarEnemigo**

Prototipo: int insertarEnemigo(MALLA \*posicion, PERSONAJE \*\*enemigos, int indice, int vidas, int n);

Descripción: En esta función se insertan los personajes que serán necesarios en el juego, cada vez que se inserte un personaje que sea diferente se tendrá que hacer un nuevo llamado a la función indicando la lista a la que se le agregarán los personajes y el índice de cual es la imagen que tendrán, al igual que sus vidas y el numero de personajes que se agregaran a la lista.

|  |  |
| --- | --- |
| PARÁMETROS | DESCRIPCIÓN |
| MALLA \*posicion | Se envía la primera posición de la malla en donde se colocara por primera vez al personaje |
| PERSONAJE \*\*enemigos | Aquí se recibe la estructura de la lista a la que se le agregaran los personajes |
| int indice | La imagen del personaje que se pondrá por primera vez |
| int vidas | Vidas del personaje |
|  |  |
| int n | Numero de personajes que se agregaran a la lista |

1. **crearEnemigo**

Prototipo: PERSONAJE \*crearEnemigo(MALLA \*posición, int indice, int vidas);

Descripción: Crea una estructura de tipo PERSONAJE que regresa para que se pueda agregar a alguna lista

|  |  |
| --- | --- |
| PARÁMETROS | DESCRIPCIÓN |
| MALLA \*posicion | Se envía la primera posición de la malla en donde se colocara por primera vez al personaje |
| int indice | La imagen del personaje que se pondrá por primera vez |
| int vidas | Vidas del personaje |
|  |  |
| int n | Numero de personajes que se agregaran a la lista |

1. **AutomatizarAlien**

Prototipo: void AutomatizarAlien(PERSONAJE \*alien, PERSONAJE \*nave);

Descripción: En el juego hay 2 tipos de enemigos, unos son los aliens y otros las naves, esta función lo que hace es automatizar cada uno de estos personajes, ya sea moviéndolos de izquierda a derecha o haciendo que disparen cada determinado numero al azar.

|  |  |
| --- | --- |
| PARÁMETROS | DESCRIPCIÓN |
| PERSONAJE \*alien | Lista que contiene los aliens enemigos |
| PERSONAJE \*nave | Lista que contiene las naves enemigas |

1. **reduceVida**

Prototipo: void reduceVida(int x,PERSONAJE \*\*alien);

Descripción: En esta función se lleva a cabo la actualización de las vidas de los enemigos, cuando se llama hace una búsqueda de el índice del alien al que se le debe reducir la vida y en caso de que éste llegue a cero vidas se elimine.

|  |  |
| --- | --- |
| PARÁMETROS | DESCRIPCIÓN |
| int x | Índice del enemigo a actualizar |
| PERSONAJE \*\*alien | Lista donde se encuentra el enemigo a actualizar |

1. **Nivel**

Prototipo: void Nivel(int n);

Descripción: Esta función sólo hace una impresión del numero de nivel en el que se encuentra el juego

|  |  |
| --- | --- |
| PARÁMETROS | DESCRIPCIÓN |
| int n | Indica el numero del nivel en el juego |

1. **elimAlien**

Prototipo: int elimAlien(PERSONAJE \*\*alien ,int x);

Descripción: Esta función hace la eliminación de un enemigo que se encuentre (por medio de la variable x) en la lista que se reciba

|  |  |
| --- | --- |
| PARÁMETROS | DESCRIPCIÓN |
| PERSONAJE \*\*alien | Lista que contiene al enemigo a eliminar |
| int x | Índice del personaje a eliminar |

1. **elimIniAtaque**

Prototipo: int elimIniAtaque(ATAQUE \*\*);

Descripción: Aquí se hace la eliminación del ataque que se detecte que esta chocando un una de las orillas de la malla o cuando el ataque toca a un adversario

|  |  |
| --- | --- |
| PARÁMETROS | DESCRIPCIÓN |
| ATAQUE \*\*a | Lista del ataque que se eliminará |

1. **visualizaImagen**

Prototipo: void visualizaImagen(MALLA \*, IMAGENES \*, int);

Descripción: Permite la visualización de una imagen que está contenida en el arreglo de imágenes, accediendo a través del índice. La visualiza en pantalla en el lugar determinado por el apuntador posición.

|  |  |
| --- | --- |
| PARÁMETROS | DESCRIPCIÓN |
| MALLA \*posicion | Apuntador hacia la malla que indica el lugar donde se visualizara la imagen. |
| IMAGENES \*imágenes | Arreglo que contiene todas las imágenes. |
| int indice | Valor que permite el acceso al arreglo de imágenes, para obtener una de ellas. |

1. **liberaPersonaje**

Prototipo: void liberaPersonaje(PERSONAJE \*\*p);

Descripción: Permite liberar el espacio utilizado en memoria por cada una de los personajes contenidos en la lista.

|  |  |
| --- | --- |
| PARÁMETROS | DESCRIPCIÓN |
| PERSONAJE \*\*p | Lista de personajes a eliminar |

1. **liberaAtaque**

Prototipo: void liberaAtaque(ATAQUE \*\*a);

Descripción: Permite liberar el espacio utilizado en memoria por cada una de los ataques contenidos en la lista.

|  |  |
| --- | --- |
| PARÁMETROS | DESCRIPCIÓN |
| ATAQUE \*\*a | Lista de ataques a eliminar |
|  |  |

1. **limpiaPantalla**

Prototipo: void limpiaPantalla();

Descripción: esta función hace un limpiado de las pantallas usadas en la paginacion

1. **limpiaPag**

Prototipo: void limpiaPag();

Descripción: Lo que hace es pintar un cuadro de color negro en el espacio del menú en donde se visualizan los records y la ayuda y así no tener que hacer un cleardevice();

1. **waitLMC**

Prototipo: void waitLMC();

Descripción: Se espera hasta que el usuario de un click izquierdo con el mouse y al final limpia el buffer del mismo.

1. **portada**

Prototipo: void portada(int x, int y, int T, int t);

Descripción: Hace una impresión de la portada del proyecto, los valores de “x” y “y” indican la posición en donde se imprimirá, la variable T indica el tamaño del titulo de la portada y la variable t indica el tamaño de todas las demás letras de la portada

|  |  |
| --- | --- |
| PARÁMETROS | DESCRIPCIÓN |
| int x | Coordenada en x donde empezará a aparecer la portada |
| int y | Coordenada en y donde empezara a aparecer la portada |
| int T | Indica el tamaño que tendrá el titulo de la portada |
| int t | Tamaño de las demás demás letras de la portada |

1. **leerNombre**

Prototipo: int leerNombre(int x, int y, RECORDS \*rec );

Descripción: imprime el lugar en donde se escribirá el nombre del jugador y al final dará un condicional de si se acepta el nombre ingresado, si se acepta regresa un 1 y la cadena en los records, y si no regresa un 0.

|  |  |
| --- | --- |
| PARÁMETROS | DESCRIPCIÓN |
| RECORDS \*rec | Estructura vacia en la que se insertara el nombre inicial del jugador |
| int x | Coordenada en x de donde empezará a imprimirse el nombre |
| int y | Coordenada en y en donde empezará a imprimirse el nombre |

1. **programa**

Prototipo: void programa(RECORDS \*rec);

Descripción: ésta es la función que lleva a cabo el algoritmo que hace el llamado a todas las funciones necesarias para el manejo del menú en el juego.

|  |  |
| --- | --- |
| PARÁMETROS | DESCRIPCIÓN |
| RECORDS \*rec); | Puntaje y nombre obtenidos del jugador |
|  |  |

1. **juego**

Prototipo: int juego(RECORDS \*rec, int nA int nN, char \*cad);

Descripción: Es la función en la que se lleva a cabo todo el movimiento del juega, ésta hace el llamado a todas las funciones para su manejo adecuado, regresa un 1 si se gano el nivel y regresa u 0 cuando las vidas del héroe llegan a 0.

|  |  |
| --- | --- |
| PARÁMETROS | DESCRIPCIÓN |
| RECORDS \*rec | Estructura que contiene los puntos y el nombre del jugador |
| int nA | Indica el numero de aliens que aparecerán en el nivel |
| int nN | Indica el numero de naves que aparecerán en el nivel |
| char \*cad | Esta función sirve para regresar una cadena con el tiempo que duró el juego |

1. **perdiste**

Prototipo: void perdiste(RECORDS \*rec ,char \*cad, int n, char \*cad2);

Descripción: Esta función sirve para indicar si el jugador perdió o ganó el juego, recibe la cadena que indica la palabra ¡PERDISTE! O ¡GANASTE! Respectivamente, imprime los datos finales del jugador (puntaje, nombre, tiempo, nivel).

|  |  |
| --- | --- |
| PARÁMETROS | DESCRIPCIÓN |
| RECORDS \*rec | Estructura que contiene el nombre y puntaje final del jugador |
| char \*cad | Contiene la frase ¡GANASTE! o ¡PERDISTE! |
| int n | Contiene el nivel final al que se llegó |
| char \*cad2 | Contiene el tiempo final del juego |

**FUNCIÓN DE SONIDO**

Para la función de sonido se implementó la función PlaySound, lo que hace es hacer reproducir un archivo indicado en su primer parámetro y en el tercer parámetro indicar el modo de reprduccion del sonido.

Para poder utilizar esta función fue necesaria la implementación de unas ligas en la sección de argumentos del programa en las opciones del proyecto y se tuvo que agregar una librería en la carpeta del programa.

**Ligas:**

-lcomdlg32

-luuid

-loleaut32

-lole32

libwinmm.a

**Librería:**

libwinmm.a

Sonidos utilizados en el juego:

**defensa.wav** – Cada vez que uno de los ataques enemigos choca con el escudo del héroe.

**Game Over.wav** – Cuando las vidas del héroe llegan a 0.

**gana.wav** – Cuando se logra pasar los 5 niveles del juego

**item** – cuando se logra obtener el indicador de las flechas

**link 7.wav** – Cuando se le decrementa una vida al héroe

**link8.wav** – Cuando el héroe muere

**menu.wav** – Cuando se encuentra en el menú principal

**nivel.wav** – Cuando se pasa a un nuevo nivel

**wchoose.wav** – Cuando se logra decrementar en 1 la vida de un alien