**OMAR SIMÓN FRANCISCO PRIETO CHACÓN 0811110010**

**GABRIEL PASTOR SANTIAGO PRIETO CHACÓN 0811110009**

**JOHN SEBASTIÁN GANTIVA DÍAZ 0811110007**

**ANTE PROYECTO DE AULA PROGRAMACIÓN EN C**

**PROFESOR: NICOLÁS MALAVER**

**QUICK LOGIC**

El nombre de nuestro proyecto de aula es “Quick Logic” y se basa en la implementación del lenguaje “C” trabajado en clase para desarrollar una programación que genere aleatoriamente laberintos dándole también la opción al usuario de seleccionar entre una variedad de íconos que serían usados como el pequeño “robot” que seguiría las órdenes del usuario para salir de este laberinto y para el diseño de las paredes del laberinto; además de poder seleccionar estos íconos, el usuario también podrá seleccionar a su gusto los colores en que se verán los íconos y el mismo fondo de la pantalla.

Se nos ocurrió esta idea ya que desde un comienzo todos nos inclinamos por intentar programar un mini juego; pero terminamos escogiendo los laberintos ya que las otras opciones se programarían de una manera muy sencilla o muy compleja, mientras que con este objetivo se pueden implementar todas las componentes que tiene el lenguaje de una manera que requiere un conocimiento promedio del lenguaje.

Además de todas las funciones del proyecto, intentaremos implementar cronómetros y puntajes que serán definidos con respecto al nivel de dificultad del laberinto que a su vez será definido con respecto al método de creación del laberinto.

Pasando a como se resolverá el problema, lo más sencillo será implementar las opciones de borrado de pantalla, repetición del proyecto al finalizar un laberinto y las opciones en las que el usuario decide todos los factores iníciales del proyecto como los íconos a utilizar. Aparte de esto, el “robot” estará programado para iniciar su recorrido en la esquina superior izquierda de la pantalla y el objetivo del juego será llegar a la esquina inferior derecha de esta a través de los comandos.

Luego de esto está la creación del laberinto en sí, para lo que tenemos pensado usar una matriz de veinte por ochenta espacios (creada a partir de arreglos de enteros) que llenaremos con datos enteros a partir de la función rand () que ingresa un número aleatorio a cada variable que se encuentra en el rango que va desde cero hasta 7FFF en notación hexadecimal o 32767 en notación decimal, que reduciremos al rango de 0 a 18 y de 0 a 79 para cada sección del arreglo del laberinto; con estas variables aleatorias llenaremos usando el método recursivo que consiste en dividir la zoina en la que se aplica en cuatro secciones y luego rompe 3 de estas estas paredes.

Después de tener implementada la creación del laberinto, nos centraremos en la programación de los comandos que el “robot” recibirá para ejecutarlos en la pantalla de una manera que lucirá como un movimiento que el “robot” hará en solo sentidos de arriba-abajo, abajo-arriba, izquierda-derecha, derecha-izquierda.

Por último deseamos desarrollar una pantalla en la que se vea el tiempo que le llevó al operador resolver el laberinto, el puntaje (que se basará en el tiempo restante), junto con el puntaje de el mismo, y alguna “imagen” que demuestre la victoria cuando se haya llegado al final del laberinto, para después dar la opción de iniciar un nuevo laberinto o repetir el que acaba de resolver donde simplemente no se aplicaría la función rand sobre la matriz y se devolvería el “robot” al punto inicial.

**FUNCIONES**

Las siguientes funciones se han desarrollado con el tiempo que llevamos programando y hemos ido necesitando en el transcurso de nuestro proceso de programación; Para la para los archivos desarrollamos las funciones de guardar perfil con puntajes, cargar datos del jugador, cargar laberintos:

|  |  |  |
| --- | --- | --- |
| **Pre condición** | **Función** | **Pos condición** |
| * Los datos de la estructura “juego->datos”, deben estar ocupados por datos válidos para poder guardarlos en el archivo deseado. * El archivo deseado y su ubicación deben estar señalados correctamente. | Guardar perfiles con puntajes | * El archivo donde se guardan los puntajes tendrá el perfil determinado en su contenido de la manera especificada de guardar por la funcion “fprintf”. |
| * la estructura “datos” debe ser pasada correctamente con los apuntadores a esta función desde el main para tener un espacio donde almacenar los 10 datos que lee con la función “fscanf”. * El archivo deseado y su ubicación deben estar señalados correctamente. | Cargar datos del jugador | * los datos de los 10 perfiles almacenados en el archivo determinado deben estar correcta y completamente almacenados en la estructura “datos”, para luego pasar los datos de un solo jugador a la estructura ”juego->datos”. |
| * Los datos de la estructura “juego->laberinto” deben haber sido pasados correctamente desde el main con los apuntadores. * Los archivos de los laberintos deben estar bien almecenados además de la ubicación de estos. | Cargar laberintos | * La estructura “juego->laberinto” debe estar llena con los datos del archivo que contenga el laberinto deseado. |

Para la interfaz gráfica desarrollamos las funciones generales de imprimir los datos de la estructura “juego->laberinto”, y los menu’s del juego:

|  |  |  |
| --- | --- | --- |
| **Pre condición** | **Función** | **Pos condición** |
| * Cada una de estas funciones necesita que lo datos de las estructuras “datos” y “juego->laberinto” les sean pasados a cada una de las funciones de los menu’s para usarlos en el momento de imprimir estos datos. | Menu’s | * Cada una de estas funciones limpia la pantalla, imprime con la función “printf” las opciones de cada menu, toma la selección con “getch” y manda la función siguiente con respecto a las opciones por medio de condicionales. |
| * La estructura de “juego->laberinto” debe ser pasada correctamente desde el main con los apuntadores de las funciones. | Imprimir laberinto | * L a pantalla es limpiada y le son re-impresos los datos char que se encuentren en la estructura “juego->laberinto ” cada vez que se llama esta función. |

Para la lógica de nuestro proyecto, desarrollamos las funciones de “mover” y de generar los laberintos aleatorios:

|  |  |  |
| --- | --- | --- |
| **Pre condición** | **Función** | **Pos condición** |
| * La estructura de “juego->laberinto” debe ser incorporada a esta función a partir de apuntadores correctamente desde el “main”. * Las bibliotecas de las diferentes funciones usadas en este segmento como “getch” y “GetTickCount” deben estar presentes y correctamente incluidas en el archivo \*.c | Moverse en el laberinto | * La función toma los valores ingresados con las flechas del teclado, los procesa y da una respuesta para que cuando llame a la función “imprimirLab”, este imprima la ilusión de movimiento de el pequeño robot. * La función retorna un valor para verificar si al final de la función “mover” se finalizó con la tecla enter (es una especie de exit en este caso)o si terminó satisfactoriamente el laberinto. |
| * La estructura de “juego->laberinto” debe ser incorporada a esta función a partir de apuntadores correctamente desde el “main”. | aleatorios | * Esta función llena las casillas de los valores correspondientes tal que pueden formar un laberinto con la función imprimirLab. |