Programador Universitario/ Lic. En Informática

Asignatura: Taller de Lenguajes I

Proyecto final de
Taller de Lenguajes I

Laberinto

Integrantes:

- **Jimenez, Silvio Martín**
- **Yurquina, Matias Exequiel**

Año 2016

Contenido

Breve manual de uso

El programa cuenta con dos listas principales de opciones que a continuación se describen:

Generar

1. Crear laberinto
   Seleccione el tamaño de laberinto que desea crear:
   I. Grande
   II. Pequeño
2. Salir

Menú “Generar”: contiene las herramientas necesarias para la creación del laberinto. Dentro de este menú se encuentran las siguientes opciones:

Opción “Crear laberinto”: El programa solicita al usuario el tamaño de la matriz a construir para realizar el laberinto. La selección se restringe a dos tamaños, como sigue:

- Grande: genera una matriz para el laberinto de dimensiones 25x80
- Pequeño: genera una matriz para el laberinto de dimensiones 20x50

Para la generación del laberinto se aplican ciertas restricciones en su construcción, que se detallaran más adelante en este trabajo. De la misma forma se tiene en cuenta el caso especial en el cual no se encuentre solución posible al laberinto.

Opción “Salir”: Permite salir del programa.

Jugar

1. Jugar
2. Top 10
3. Salir

Menú “Jugar”: contiene las opciones siguientes:

Opción “Jugar”: Otorga acceso a la lista de laberintos guardados para, consecuentemente, ejecutar el seleccionado.

Opción “Top 10”: Direcciona al Ranking de posiciones de los 10 mejores puntajes logrados (considerando el de menos pasos como el mejor).

Opción “Salir”: Permite salir del programa.

Crear Laberinto

Una vez seleccionada la dimensión del Laberinto (Grande o Pequeña), el programa muestra por pantalla el Laberinto sin senderos (todo pared), con un Inicio (marcado con #) y un final (marcado con @) predeterminados. El puntero se ubica en la esquina superior izquierda del Laberinto y en un principio se encuentra activa la opción de moverse sobre en el interior del Laberinto. Ésta aplicación cuenta con las siguientes teclas de función para hacer las distintas modificaciones sobre el mapa:

- **E –** Activa la función de moverse sobre el mapa, sin hacer modificaciones.
- **S –** Activa la función de dibujar senderos en el mapa.
- **R –** Activa la función de rellenar con paredes en el mapa.
- **B –** Rellena todo el mapa con paredes, como en su estado inicial.
- **I –** Activa la función para marcar el Inicio del Laberinto (esta función espera un ENTER para desactivarse).
- **F –** Activa la función para marcar el Final del Laberinto (esta función espera un ENTER para desactivarse).
- **G –** Con ésta tecla se finaliza la creación del Laberinto y se pasa al modo prueba para verificar que encuentre una solución para el mismo.
- **ESC –** Con la tecla escape se sale de la edición del mapa, y se regresa al menú de Generar.

Una vez generado el Laberinto, y luego de que haya pasado el modo de prueba, el programa guarda el archivo binario que contiene el Laberinto y le pregunta al usuario si desea crear otro laberinto. En caso afirmativo vuelve al menú de Generar y en caso contrario se Cierra.

Jugar

En el Menú de jugar están las opciones de ver el Top10, que muestra el Ranking con los 10 mejores puntajes obtenidos y el nombre de los jugadores que lo lograron; Y por otro lado la opción de Jugar alguno de los laberintos creados. En ésta última opción se despliega una lista con los Laberintos creados y espera a que se seleccione alguno para jugar. Una vez que se elige un laberinto, lo abre y el jugador puede comenzar a moverse dentro de él. En ésta pantalla el jugador puede presionar la tecla **ESC** para volver al menú de Laberintos. También se muestra por pantalla la cantidad de pasos que va realizando el jugador hasta llegar al final.

Cuando el jugador gana, se muestra en la pantalla el mensaje “Ganaste” dentro de un recuadro y se le indica su puntaje. Si éste puntaje cumple con la condición de estar dentro del Top10, entonces se le pide el nombre al jugador y se guarda su puntaje en el Top10, caso contrario se lo felicita por su puntaje y se muestra un mensaje.

Finalmente se le ofrece al usuario seguir jugando otro Laberinto o salir.

Estructura de la cabecera del archivo binario

La estructura de la cabecera del archivo binario que almacena el laberinto (en forma de matriz binaria), tiene la siguiente forma:

- Un campo **name[17]**, cadena de caracteres que ocupa **17 bytes** de memoria y almacena el nombre con el que fue guardado el Laberinto. Que consta de la palabra “Laberinto” seguida de la numeración correspondiente (en 3 dígitos), y por último la extensión “.dat”.

- Un número entero **m**, que ocupa **4 bytes** de memoria, y almacena la dimensión de altura del Laberinto.

- Un número entero **n**, que ocupa **4 bytes** de memoria, y almacena la dimensión del ancho del Laberinto.

- Un número entero **x**, que ocupa **4 bytes** de memoria, y almacena la posición de la columna en la que se encuentra el inicio del Laberinto.

- Un número entero **y**, que ocupa **4 bytes** de memoria, y almacena la posición de la fila en la que se encuentra el inicio del Laberinto.

En total la cabecera ocupa **33 bytes** de memoria en el archivo binario.

Descripción de cada una de las funciones creadas

Void gotoxy(int x, int y):

Función void, recibe dos parámetros enteros, x e y, luego posiciona el cursor en el lugar que éstos parámetros indiquen. Los “x” positivos mueven el cursor hacia la derecha, y los negativos hacia la izquierda mientras sea posible. Los “y” positivos mueven el cursor hacia abajo y los negativos hacia arriba mientras sea posible.

Bool contorno(int m, int n, int i, int j):

Devuelve un valor booleano, recibe como parámetros las dimensiones de la matriz y las coordenadas del cursor i y j. Revisa que el cursor no se haya posicionado en alguna esquina, ni en otro lugar que no sea el contorno de la matriz. Devuelve “true” si el cursor se encuentra en el contorno y “false” si el cursor se encuentra en el interior de la matriz.

Int mover(int m, int n, int *i, int *j, int contorno):

Funcion int que devuelve el valor de “x” obtenido según la tecla que es presionada. Es usada para mover el cursor de coordenadas (i,j) dentro de los límites fijados por las dimensiones “m” y “n” de la matriz, y el valor entero de contorno, usado como true (1) o false (0).

Si “x” obtiene un 0, se evalua el segundo valor entero obtenido por la tecla presionada con una variable “y” que indicará si cual fue la tecla de movimiento presionada y llamando a la función “gotoxy” moverá el cursor usando como parámetros, los valores “i” y “j” enviados a la función por referencia e incrementados o no según el movimiento indicado.

Si contorno es 0, los límites para moverse son dentro de la matriz, sin contar los contornos. En cambio si contorno es 1, permite también moverse por el contorno.

Int senderos(int m, int n, int *i, int *j, int **Mat):

Funcion int que devuelve el valor de x modificado por la función “mover”. Esta función dibuja los senderos del laberinto (llamando a la función mover()) y a la vez le asigna un 0 a la matriz dinámica en las posiciones (i,j) que vaya dibujando el sendero. Esta función es invocada cuando el programa detecta que se presionó la tecla “S” (mayúscula o minúscula). Y termina su funcionamiento cuando se presiona una tecla que indique otra función.

Int noEscribir(int m, int n, int *i, int *j):

Función int que se invoca cuando el programa detecta que se presiono una “E” (mayúscula o minúscula), mueve el cursor sin realizar cambios en la matriz, atraves de la función mover que va modificando el valor de “x” según la tecla que se presione. Realiza esto hasta que “x” toma el valor de alguna tecla que llama a otra función y entonces retorna la “x”.

Int reescribir(int m, int n, int *i, int *j, int **Mat):

Función int que trabaja de la misma manera que la función senderos(), con la diferencia de que escribe 1 en la posiciones de la matriz por donde se vaya moviendo el cursor. Se usa para reescribir muros en donde haya un sendero. Retorna el valor de “x” cuando se presione una tecla que active otra función.

Void *ReservarMemoria(int TotalBytes):

Reserva tanta memoria en Bytes, en tiempo de ejecución, como se le indique en el único parámetro.

bool obtenerInicio(int **Mat, int m, int n, int *x_p, int *y_p):

Función Booleana que recorre toda la Matriz buscando la posición de inicio del Laberinto. Si encuentra dicha posición, entonces asigna las coordenadas de esa posición en las variables “x_p” e “y_p” que fueron enviadas como parámetros por referencia a la función y luego retorna true. En caso de no encontrar la posición de inicio, retorna false.

Int ** asignar_matriz(int m, int n):

Función que devuelve un puntero doble, que contiene la reserva de memoria para la Matriz de dimensión mxn.

Char ** asignarChar(int m):

Esta función trabaja de la misma manera que la anterior, con la diferencia de que hace una reserva para m cadenas de texto de longitud 17. Y retorna el puntero doble a char. Es usada para guardar los nombres de los laberintos creados a medida que se los va leyendo de la carpeta de fuente.

void genMat(int **Mat, int m, int n):

Esta función que no tiene retorno, lo que hace es rellenar con “1” la Matriz de enteros que se le envía por parámetro como puntero doble, de dimensiones mxn (enviadas como parámetro).

Void mostrarMatriz(int **Mat, int m, int n, bool jugando):

Esta función void, muestra la Matriz de enteros que se le envía como doble puntero, de dimensiones mxn. Según el valor booleano que se le envía en la variable “jugando” muestra la matriz para jugar (paredes y personaje) o para seguir editándola.

int moverMenu(int *i, int *j, int lim_sup, int lim_inf, bool uno):

Esta función lo que hace es mover un índice en un menú, es similar a la función mover, con la diferencia de que solo admite movimientos hacia arriba o abajo para desplazarse entre las opciones de un Menú, cuyos límites de movimiento se le envían por parámetro en la función. El valor booleano que recibe en la variable “uno” indica si el índice se moverá de a una o de a dos posiciones.

void jugar(int m, int n, int x_p, int y_p, int **Mat, bool *win, bool jugar, int *pasos):

Función que recibe como parámetros la dimensiones de la matriz (mxn), las coordenadas del inicio del laberinto, la Matriz que contiene el laberinto, dos valores booleanos: “win” que enviado por referencia inicialmente en false, y “jugar” que indica si está jugando el juego o está haciendo la prueba en el Generador de Laberintos, y por último un valor entero pasos enviado por referencia que servirá de contador de pasos. En ésta función se hace el movimiento del jugador sobre la matriz, en la que se controla que solo se mueva por donde haya senderos y mientras tanto vaya sumando el contador de pasos en 1 por movimiento. En caso de llegar al final, convierte el valor de “win” en true, y termina el ciclo.

void nombreLaberinto(int n, char name[]):

Recibe un valor entero “n”, y un arreglo de caracteres “name”. Lo que hace es convertir el numero entero en un arreglo que contenga el mismo número pero como carácter. Luego concatena este arreglo generado con el “name” enviado a la función, y por último a la cadena resultante de estos dos, le concatena “.dat”. De ésta manera forma el nombre con el que se guardará o desde el que se buscará un archivo binario.

int cantidadListas():

Esta función devuelve la cantidad de Laberintos que hay creados hasta el momento, más uno (debido a que el contador se inicializa en 1). La manera de contar la cantidad de Laberintos a controlar si se puede abrir los archivos de nombre “Laberinto” seguido de la numeración indicada. Si lo puede abrir, entonces suma el contador y busca el siguiente laberinto.

char **obtenerLista(int m):

Esta función recibe como parámetro un entero “m”, y devuelve una lista de “m” cadenas de caracteres de longitud 17, como puntero doble de char. Hace uso de las funciones asignarChar() y nombreLaberinto().

void guardarMatriz(int **Mat, int m, int n, int x_p, int y_p):

Esta función void se activa cuando el programa detecta que se presionó la letra “G” (mayúscula o minúscula). Lo que hace es generar un archivo binario con el nombre del nuevo Laberinto, usando la función nombreLaberinto(), y guardando en él la Matriz que ya fue modificada durante la ejecución del Generador de Laberintos. Tambien guarda al final del archivo la Cabecera del mismo, en la que se incluye el nombre, las dimensiones y la posición del inicio y final del laberinto.

void LeerDeCabecera(int * m, int * n, int * x_p, int * y_p, char name[], char Lista[]):

Ésta función lo que hace es Leer los datos de la cabecera del archivo cuyo nombre se especifica en Lista[] (enviado como parámetro), y almacenarlos en las restantes variables enviadas a la función por referencia para que las mismas sean modificadas en su posición de memoria y luego puedan ser usadas.

void leerMatriz(int **Mat, int m, int n, char name[]):

La tarea de ésta función void es almacenar en la Matriz que se le envía como parámetro atraves de un puntero doble, los valores de la Matriz que está almacenada en el archivo cuyo nombre se indica en el arreglo name[] enviado como parámetro a la función.

void guardarTopTen(char Mat[10][11], int pasos[], int nuevo, char player[10]):

Recibe como parámetros: un arreglo de 10 cadenas de caracteres con longitud 11, un arreglo de enteros que contiene los 10 mejores puntajes de laberintos, el nuevo puntaje que debe entrar al Top10, y el nombre del jugador que lo logró. Lo que hace es abrir el archivo binario donde se guardan los puntajes en modo escritura, de manera que se reescriba el existente o en caso contrario lo cree. Luego, controlar la posición del arreglo pasos[] en la que debe ir el nuevo puntaje logrado, agregarlo en esa posición y luego mover todos los otros puntajes una posición abajo en el Top10, junto con sus nombres. El último puntaje en moverse, que pasa a ser el peor, no se guarda. Finalmente modificados el vector de Nombres y Pasos, se reescribe esos valores en el archivo binario de puntajes.

void ordenarPasos(int vPasos[], char Mat[10][11]):

Ésta función recibe como parámetros el vector de pasos, y el vector de nombres de jugadores del Top10. Lo que hace es ordenarlo en orden ascendente, a través de un método de ordenación estudiado en la carrera, muy eficiente.

void leerTopTen(char Mat[10][11], int pasos[]):

Ésta función guarda en un vector de 10 nombres y un vector de 10 enteros, los nombres y puntajes leídos del archivo binario del Top10, en su primera invocación predeterminará el archivo de puntajes con nombres “No name” y puntaje “999”.

Conclusiones

Luego del trabajo realizado durante estas semanas, estas son algunas de las conclusiones que pudimos obtener:

- El trabajo en equipo es muy provechoso a la hora de tomar decisiones, elegir lo mejor, proponer, dividir el trabajo y sirve mucho ampliar el conocimiento, complementando los de cada integrante.
- A la hora de programar es conveniente el uso de la metodología Divide&Conquer. Una vez dividido el trabajo, ir realizándolo uno a la vez y probar que funcionen por separado los procesos, manteniendo una relación de sintaxis y/o nomenclaturas entre ellos para que luego sea más fácil su unión.
  Y finalmente, realizar las uniones de los procesos realizando pruebas parciales de su funcionamiento en conjunto.
- Si se trabaja en archivos por separado, y éstos comparten algunos procesos, es útil armar un archivo de cabecera, que contenga las declaraciones y definiciones de estas funciones y luego incluirla en ambos archivos. De ésta manera se reduce líneas de código y se logra estandarizar una función para distintos usos.
- Cuando se trata de un juego, es muy importante analizar en un principio todas las opciones y/o los casos que se pueden presentar durante la ejecución del mismo. Ya que si se deja de lado algún detalle el juego se puede detener en alguna instancia y el producto no sería bueno.
- El diseño en cuanto a lo estético ayuda mucho a la impresión que se lleva un usuario al ver una aplicación. Por eso, es recomendable usar colores con un buen contraste, destacar las partes importantes, y poner énfasis en las secciones donde el usuario podrá intervenir para que éste se vea motivado a hacer uso de las mismas.
- En algunos casos, es conveniente estandarizar o fijar opciones para el usuario y no dejar todo en sus manos. Ya que éste puede no tener un buen conocimiento de cómo funciona el programa en su interior y es posible de que de alguna indicación que ocasione errores en el sistema.
- Los mensajes por pantalla son necesarios para que el usuario entienda que es lo que el programa espera que haga en ese momento, o cual indicación está esperando para continuar con su ejecución.

Bibliografía y Webgrafía

- Material de Clases de la Asignatura Taller de Lenguajes I.
- Material de Clases de la Asignatura Algoritmos y Estructura de Datos.
- Páginas de internet, sugeridas por Google, relacionadas con C++, sus funciones y librerías.