***DOCUMENTACION PUZZLE 15 – Valentino Privitera***

**Analista Universitario en Sistemas**

**Taller de Programación I**

Lic. Luciano Diamand

Lic. Diego A. Bottallo

* **Indice**

**- Introduccion: Presentación del Juego**

**- Desarrollo en C: Funciones de C empleadas para la realización del juego**

**- Funciones y Algoritmos: Diseño e implementación de los algoritmos**

**- Librería: “BaseDatosMatrices.h”**

**- Compilación y Ejecución**

* **Introduccion**

El Puzzle 15, o Solitario del 15, fue creado en el siglo XIX y es un juego de deslizamiento, donde en una grilla de dimensiones 4x4, se disponen aleatoriamente 15 piezas numeradas, sin repetir, del 1 al 15, quedando un espacio vacío.

Se parte de una configuración aleatoria (foto de la izquierda) y el objetivo del juego (solución) es ordenar dichas piezas logrando una disposición final (foto de la derecha).

El espacio vacío sólo puede ocuparse por una de las fichas vecinas o adyacentes, que comparten una coordenada horizontal o vertical (no una vecina en una posición diagonal), es decir, si observamos la figura de la izquierda, el espacio vacío en la posición (fila3, columna4), sólo puede ocuparse por las piezas 14, 7 o 6 (generando un nuevo y único espacio vacío al desplazarse), pero no ocuparse por las piezas 10 o 12 (diagonales) ni cualquier otra.

Vamos a enumerar los requerimientos de este juego, para trabajarlos y desarrollarlos a lo largo de la documentación. En cada sección se comenta una parte del juego y la función que corresponde a cada una.

* **Desarrollo en C:**

En el siguiente apartado vamos a enumerar las funciones y herramientas que nos permitieron desarrollar nuestro programa. Las mismas las obtuve de las librerías **<stdio.h>**, **<stdlib.h>** y **<ctype.h>**.

Para diseñar el tablero de juego, creamos una matriz con un **array** **bidimensional** de 16 elementos en forma de 4 filas (eje Y) y 4 columnas (eje X) donde se alojarán los números. El objetivo es que se genere una configuración solucionable cada vez que se inicie una partida.

Mediante el uso de estructuras **Switch** se diseñó la base de datos de matrices resolvibles la cual veremos en detalle mas adelante, y se aplicó también a la consola de juegos, ósea, la función que permite el movimiento dentro del tablero.

Además, se hizo múltiples usos de estructuras repetitivas como son el **Do While** para el flujo del juego y el **For**, para poder leer y modificar estas matrices a gusto.

Obviamente el uso de **printf**, **getchar()** y **toupper()** para las interacciones del programa con el usuario.

Y, por último, pero no menos importante, el uso de **funciones** creadas por el desarrollador, las cuales fueron muy útiles para crear un programa efectivo e iterativo.

* **Funciones y Algoritmos**

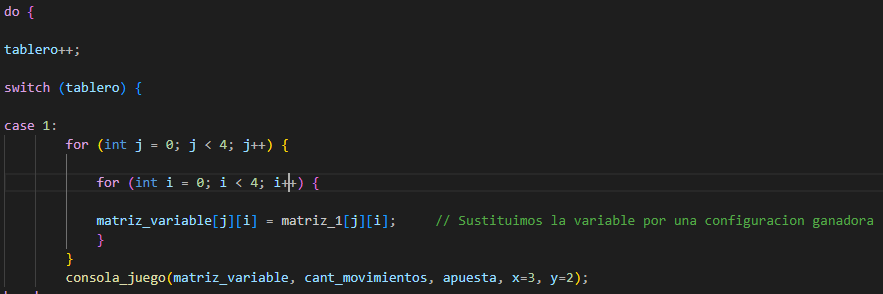
En el siguiente apartado vamos a enumerar y detallar las funciones y algoritmos diseñadas para el funcionamiento de este juego, en orden de importancia.

**- Función Main**

En el apartado de la función **main** desarrollamos un set de variables que van a ser de utilidad a lo largo del programa. Declaramos una **matriz\_variable** para que, en cada partida, se transforme en una **matriz resolvible** (esto se verá en detalle en el apartado de matrices).

Además, usaremos variables para su posterior pasaje de parámetro a funciones. Estas variables son: **cant\_movimientos** para contar los movimientos del jugador; **apuesta** para guardar la apuesta del jugador antes de empezar; y por último declaramos a **X** e **Y**, que van a indicar donde esta el 0 en cada configuración de matriz para comenzar desde ahí la ficha vacía y, por ende, los movimientos.

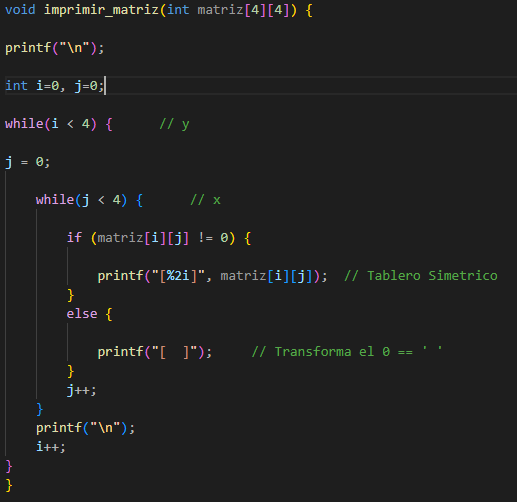
En la función **main** generamos una estructura de repetición **Do While** para que se repita el inicio de partida un **máximo de 10 veces**, y dentro de esta estructura creamos una estructura condicional **Switch**, la cual tomara de condición a la variable **tablero**, que a su vez se incrementara en 1 por vez que se inicie una partida nueva.

Ahora, dentro del **Switch**, se generan **10 cases** (para 10 partidas), y en cada case se realiza el siguiente algoritmo de transformación con estructuras repetitivas **For**. A continuación, el algoritmo en cuestión:

Generamos un **For** para las filas **(j)** y otro para las columnas **(i)**. El algoritmo consiste en suplantar **cada elemento** de la **matriz\_variable** (que es la que representa cada configuración en cada partida y se aloja en memoria), con los elementos de la **matriz\_1**, ya definida en el **header** “**BaseDatosMatrices.h**”, la cual ya posee una configuración de puzzle resolvible.

La finalidad total de la **main** es que, por partida iniciada, se presente de a uno hasta diez **case**, ósea, partidas. Estos por simplicidad, se realizan en orden consecutivo, no por aleatoriedad.

Por último, se llama a la función que se encarga del movimiento del tablero y básicamente el alma del programa, la función **consola\_juego()**.



**- Función Imprimir Matriz**

Antes de detallar a **consola\_juego()**, es preciso que explique el funcionamiento de la función **imprimir\_matriz()**, la cual nos ayuda a imprimir la matriz actual cuando deseemos. Su algoritmo es sencillo, simplemente se crean dos **For** para recorrer las filas y columnas de la matriz en cuestión, y mientras sus elementos sean distintos de 0, se va a imprimir su contenido mediante un **printf**, el cual incluye 2 espacios por corchete, para darle simetría a la representación del tablero. En caso de que detecte el 0, lo va a **sustituir** por un **espacio vacío** para crear la ficha vacía.

**- Función Consola Juego**

A continuación, vamos a explicar en detalle en que consiste la función que nos permite desplazaros a través del tablero. Esta función recibe nada mas y nada menos que 5 parámetros distintos:

El primer parámetro es **matriz [4] [4]**, el cual va a recibir una configuración de matriz distinta cada vez que se lo llame, y esta configuración va a ser dada por **matriz\_variable**. Mediante la función **imprimir\_matriz()** se dispone gráficamente el tablero de esa partida.

Dos parámetros muy importantes son **X** e **Y**, a los cuales se les pasa el valor de referencia desde la **main**, indicando en que posición se ubica el 0, para poder comenzar a mover la ficha desde ahí. Son elegidos parámetros puesto que cada matriz posee su propia disposición de 0, y esta es indicada por **X** e **Y**.

Los dos últimos parámetros son **cant\_mov** y **bet,** los cuales van a almacenar los movimientos realizados y la apuesta por partida, para su posterior uso en la obtención de los puntos del jugador.

**- Algoritmo del Movimiento**

El algoritmo que realiza los movimientos dentro del tablero se basa principalmente de una estructura condicional **Switch**, la cual obtiene un **char** mediante la función **getchar()** y dependiendo del mismo, se dan 5 **cases** distintos: **W** (Arriba), **A** (Izquierda), **S** (Abajo), **D** (Derecha), y la opción de salir de la partida con el **case** E (Exit) (esta función esta encerrada en un **Do While** y se ejecutara hasta que el usuario termine la partida o presione E).

Antes de jugar se procede a mostrar los controles del juego, se dispone el tablero de la partida actual y se le pregunta al jugador en cuantos movimientos apuesta que va a resolver el puzzle (forzamos a que la apuesta se encuentre entre 40 y 200 con un **Do While**).

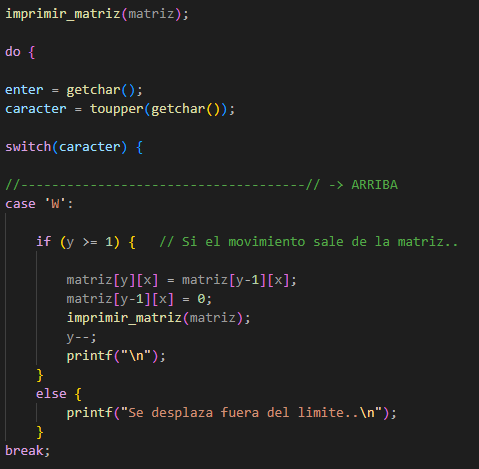
Como los **array** empiezan a contar el primer elemento desde 0, vamos a movernos en filas (Y = 0, 1, 2, 3) y columnas (X = 0, 1, 2, 3), con movimientos de a una unidad por acción (X++, Y++, X--, Y--).

Recordemos que se pasaron los parámetros de **X** e **Y** para posicionar el movimiento inicial en la ficha donde se encuentre el 0 (más abajo encontraremos una imagen que muestra todo esto).

En la próxima pagina se procede a explicar el mecanismo de movimiento.

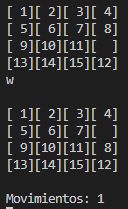
**- Mecanismo de movimiento entre celdas**

Ya explicado el algoritmo a grandes rasgos, procedemos a explicar que sucede cuando el usuario presiona uno de los caracteres aceptados (W, A, S o D).



La idea principal es que cada **case** detecte que movimiento desea realizar. En este caso, estamos presionando ‘W’, lo que significa que queremos desplazar nuestra ficha vacía hacia arriba, intercambiándola por el elemento que este en la posición adyacente superior a nuestra ficha. Mediante un condicional **If**, se evalúa si la ficha no se esta moviendo fuera de los límites.

Si quisiéramos ir hacia arriba, se evalúa que Y no sea menor a 0 (siendo 0 la primera fila), dado que nos estaríamos desplazando fuera de la matriz. En caso de que se cumpla con la condición, se procede a cambiar la posición del 0 con el elemento que se encuentre en la dirección seleccionada (una fila más arriba), y ese elemento se sobrescribe por 0. Después, donde se encontraba el 0, se sustituye por el elemento que le cambio la posición, y así es como se consigue una transición limpia.

**Por ejemplo**, en este caso, si estamos ubicados en la y=2 (tercera fila), y quisiéramos ir hacia arriba, el algoritmo detecta que y=2 pasa a ser y=1, y como cumple con la condición, se resuelve el cuerpo del **If**. Mientras el movimiento sea vertical, x va a permanecer estático. En este caso, x=3 (columna 4) y, después del movimiento, permanece en x=3;

Si quisiéramos movernos afuera de los límites del tablero, no se cumple la condición por lo tanto procede a mostrar el mensaje **Default** que alerta que el movimiento no es válido.

Cada **case** tiene delimitados sus movimientos adentro del tablero, y este control se realiza mediante un condicional **If.**

Por último, cabe resaltar que, en cada iteración del **Switch**, que no sea E, el contador **cant\_mov** se incrementa en uno, para llevar el recuento de movimientos en esa partida, y la apuesta **bet**, se almacena para su posterior evaluación y asignación de puntos.

Ahora que se detalló el algoritmo y el mecanismo de movimiento, podemos pasar al final del Switch, en donde se llama a la función **compara\_matriz\_ideal(),** la cual se invoca en cada movimiento realizado para evaluar si la configuración que va logrando el usuario es la configuración ganadora (ósea los números del 1 al 15, con el 0 al final).

Antes de desarrollar esta función, se pasan como parámetros la cantidad de movimientos y la apuesta de esa partida, para además evaluar los puntos dentro de esa función.

**- “*Matriz Ejemplar*”**

Diseñe “una *matriz ejemplar*” como **variable global**, la cual es una matriz de 4x4 que contiene la configuración ganadora. Dentro de la función **compara\_matriz\_ideal(),** esta será la que **verifique en cada movimiento** del usuario si la matriz esta completa o no. Una vez completada, se cuenta el numero **m** de movimientos que hizo el usuario, y se compara con la **apuesta** inicial que hizo antes de empezar la partida para asignarle o no los puntos.

**- Función Compara Matriz Ideal**

La función **compara\_matriz\_ideal()** posee 3 parámetros, estos son la matriz que se venía evaluando en **consola\_juego()**, los movimientos que se realizaron para completar la matriz en cuestión (**cant\_mov**), y la apuesta del jugador para esa partida (**bet**).

Mediante un algoritmo de comparación de elementos, se compara en cada movimiento la matriz del jugador con la *matriz ideal*. Declaramos una variable para indicar si ambas matrices son iguales o no, y mediante el uso de dos **For** (fila y columna) se evalúa cada elemento.

Si ningún elemento de la matriz es distinto a los elementos de la *matriz ideal*, quiere decir que las matrices son idénticas, y devuelve un 1, ósea verdadero. En caso de que algún elemento si difiera, se procede a romper ambos ciclos con la orden **break** y se devuelve 0, el cual procede a seguir con los movimientos hasta que se de con la configuración ganadora.

En caso de devolver un 1, y que ambas matrices sean iguales, se procede a finalizar la partida, notificar al jugador y a evaluar los movimientos con los puntos. Esta evaluación la veremos en el apartado de apuesta.

**- Apuesta**

Se agrego una instancia en la cual el jugador ve el puzzle, apuesta en cuantos movimientos lo puede resolver, y almacena esta apuesta para compararla con los movimientos y así otorgar puntos. Los puntos se asignan de esta forma: (m = apuesta).

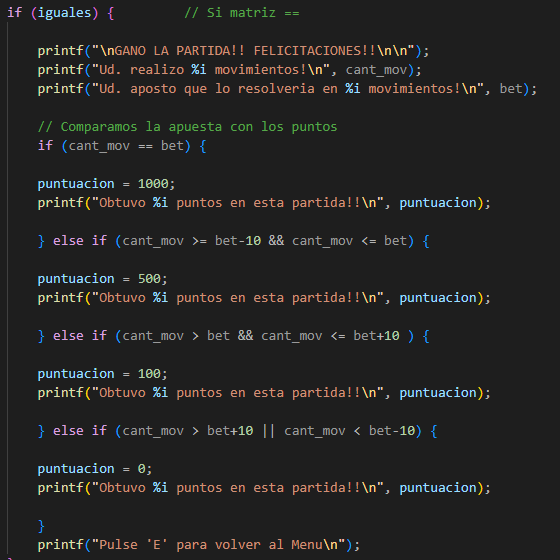
1000 puntos si lo hace en exactamente M movimientos

500 puntos si lo hace en una cantidad de movimientos dentro del rango [M-10, M]

100 puntos si lo logra en (M, M+10] movimientos

0 puntos en otro caso

Mediante varios condicionales **If** **Else**, evaluamos la situación del jugador, y asignamos los puntos con la variable **puntuación**. El algoritmo que detecta la apuesta se detalla a continuación:

­

Una vez finalizada la asignación de puntos, se devuelven al jugador todas las estadísticas de la partida que acaba de finalizar, y mediante un cartel recuerda volver al menú presionando E. Una vez volvemos al menú, nos consulta si queremos seguir jugando. Si la respuesta es S (Si), se presenta el siguiente tablero y todas las instrucciones de vuelta. Si la respuesta es N (No), se termina el juego.

* **Librería “BaseDatosMatriz.h”**

Se diseño en la **función main** una “base de datos” de matrices resolvibles, ya verificadas con solución, las cuales se almacenan en un numero del 1 al 10. Mediante una estructura condicional **Switch**, se va a ir actualizando la matriz a medida que se inicie una partida. Empieza con la matriz\_1 (case 1) y cuando se termina la partida continua con la matriz\_2 (case 2).

Como ya está definida la matriz de antemano, le indicamos en cada case, dependiendo donde se posicione el 0 en esa matriz, mediante las coordenadas **X** e **Y**, donde debe posicionarse la ficha inicial, ósea, donde este el 0, y lo suplanta con un espacio vacío.

Se almacenaron 10 configuraciones distintas, resolvibles, en un archivo **header.h** del cual se sustraerán las matrices resolvibles, y se sustituirán en la **main** por **matriz\_variable**.

* **Compilación y Ejecución**

El programa fue desarrollado en Windows, usando el IDE **Visual Studio Code**, y compilando mediante los plugins de C en el mismo editor de texto. Se advierte de 3 **warnings**, los cuales me detectan las variables **enter** como sin uso, aunque yo las declare y uso para que no me tome el **enter** como un char. El ultimo warning se debe a que detecta al parámetro **cant\_mov** sin uso, sin embargo, no generan problemas ni bugs.

El programa fue arduamente probado y se constató que no haya errores o eventos indeseados, por lo que debería funcionar correctamente.

El método usado por mi fue compilar directamente desde el IDE, y desde ahí mismo se juega por terminal, ingresando solo **números** y **char**, de la manera mas sencilla y robusta que encontré.

Si se desea salir de la partida, presione E y le va a preguntar si quiere jugar otra partida. En caso de que no, presione N y se termina el programa.