Universidad de Antioquia - Facultad de Ingeniería



Parcial II

Bohórquez Ardila, Luisa María - Morón Acacio, David Alejandro

Informática II

Docentes a cargo: Guerra Soler, Aníbal José - Salazar Jiménez, Augusto Enrique

Medellín, Antioquia. 2023

1. **Contextualización del problema:**

Se tiene para la presente entrega el programar un Othello, juego de estrategia de mesa para dos personas, escrito en el lenguaje C++ direccionado a la POO, teniendo en cuenta la eficiencia del programa y el correcto uso de la memoria, así como el manejo de excepciones y la documentación adecuada del programa. Entonces, el reto consiste en diseñar un tablero de 8x8 (donde se muestre todo el tiempo el estado de todas las fichas), cuyas posiciones se van actualizando según las fichas que jueguen los dos jugadores que estén involucrados en el juego. También, debe ser posible guardar un historial de partidas en un archivo de texto plano, con detalles sobre la fecha y la hora, el ganador y las cifras que el ganador logró.

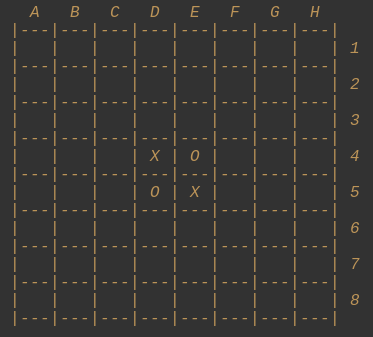
1. **Consideraciones a tener en cuenta:**

Se debe desplegar un cuadro de texto que muestre un logo de “Othello”, posteriormente cuando se acabe una partida se podrá decidir si jugar otra vez o no, por tanto el juego entero debe estar en un loop que se detenga sólo cuando una partida termina y algún usuario decida no jugar otra partida. Además del cuadro de texto con el logo, se tiene que imprimir por pantalla el tablero de juego (el cual compondremos a base de guiones y barras que den la ilusión de casillas) acompañado de las coordenadas laterales y verticales (para que los jugadores puedan elegir dónde poner su ficha), de manera que sea visible y claro para el usuario lo que está pasando. También. al momento de comenzar el juego, ambos jugadores podrán registrar su nombre y se darán los colores de las fichas a cada uno (en cuanto se registren, el programa será quien asigne cuál juega con fichas azules y cuál juega con fichas rojas por orden de llegada, siendo respectivamente guiones y asteriscos). Por nuestra comodidad, cambiamos guiones y asteriscos por letras X y O, también se usaron distintos colores en las salidas por consola para lograr una interfaz más amigable.

Teniendo en cuenta todas las validaciones de los movimientos y posiciones legales el el juego, así como las validaciones de entrada por consola, sólo resta tener en cuenta la disposición del histórico de jugadas, donde se guardan los nombres de los jugadores, las fichas que acumularon y quién ganó, todo en un archivo de texto que debe estar actualizándose cada vez que se juega. En el archivo, por cada jugada se irá añadiendo un bloque de texto en el que se registren los datos por partida, el formato se debe dar como:

| OTHELLO  Fecha y hora: //aquí se registra la fecha y la hora de inicio de la partida  Ganó “nombrejugador” con “x” fichas  …//las partidas posteriores |
| --- |

Dado que todo va a construirse a base de arreglos de arreglos, memoria dinámica, clases y librerías con manejo de excepciones y archivos de texto, se tendrán en cuenta los headers donde se guarden las funciones de mostrar el “tablero”, cambiar el contenido del “tablero” (el objeto que modela todos los arreglos que contienen los caracteres que van cambiando según la jugada), poder recibir las jugadas y poder verificarlas, validaciones, entre otros auxiliares, y en una función iría todo lo referente a las invocaciones de dichas funciones y el menú del juego, para que los objetos por sí mismos sean los que modelen el juego y que no dependa enteramente de la aplicación de un main. El tablero se formaría con arreglos que sólo posean los caracteres jugados en las posiciones indicadas, y a medida que se va imprimiendo el tablero, se van concatenando guiones, barras y espacios en blanco para lograr la interfaz que queremos, mostrando algo parecido a la siguiente figura:



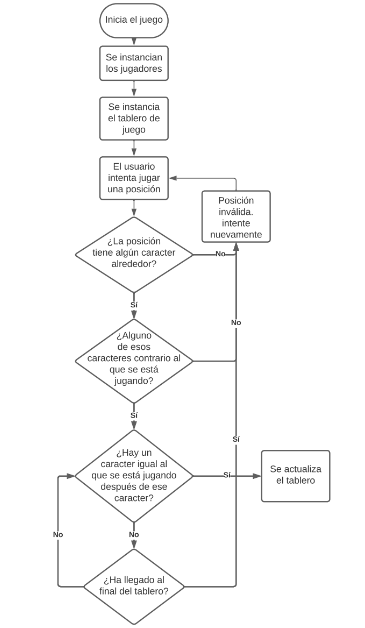
1. **Diseño de funciones, clases y algoritmos:**

Vamos a definir dos clases:

1. Clase Jugador, cuyos atributos serán el tipo de ficha (es decir, el caracter que van a estar jugando), el nombre del jugador (ingresado por el usuario) y el estado de victoria (si al final de la partida ganó o perdió), y cuyos métodos serán el recibir el nombre que ingrese el usuario, definir el caracter que está representando a su personaje, y actualizar la variable que verifica si ganó o perdió (recibiendo al final de la partida el número de fichas acumuladas).
2. Clase Tablero, cuyo atributo principal será una matriz que almacene todo el tablero, y cuyos métodos a grandes rasgos sean actualizar la matriz (de manera que, cada vez que un jugador ingrese una jugada, el método se encargue de cambiar los caracteres necesarios de la matriz, y de indicarle al usuario que es un movimiento inválido en caso de que no pueda hacer nada), un método que imprima en pantalla la matriz para indicar el estado del juego, y un método que contabilice los caracteres al final de cada partida.

De esta manera, sólo se deberán leer los atributos de las instancias de las clases para saber quién ganó y con cuantas fichas.

Entonces el programa comienza, se inicializan las clases, se pide al usuario que ingrese su nombre y se comienza a definir el objeto, empieza el juego, luego se comienza el ciclo en el que el juego se dedica a evaluar los elementos que hay en las posiciones de los arreglos alrededor de las posiciones ingresadas por el usuario para verificar si cambia un caracter, o si no puede poner una ficha en esa posición. Entonces se evalúan las posiciones por debajo, por arriba, por derecha, por izquierda, por la diagonal inferior izquierda, diagonal inferior derecha, diagonal superior derecha y diagonal superior izquierda. Nuestro algoritmo va a basarse en el siguiente diagrama:



Es decir, se va a buscar posición por posición siempre y cuando hayan fichas contrarias, hasta llegar a una posición vacía o al final del tablero. De esta forma, se garantizan las jugadas válidas y al mismo tiempo se realizan dichas jugadas, se actualizan los objetos y se guardan para ser imprimidos en pantalla. Al terminar, la partida se guarda en el registro en el archivo de texto en el formato que está indicado, también en tiempo de ejecución, para que se puedan jugar múltiples partidas continuas y que cada una se guarde en el archivo con los datos especificados (es decir, que no deba cerrarse el programa para tener que guardar las partidas, sino que se vaya escribiendo el archivo al terminar cada partida).

**Funciones a diseñar:**

***Para los archivos:***

**void ImprimirNombre(string nombre\_archivo);**

/\* Esta funcion recibo el nombre de un archivo e imprime linea por linea lo que hay en él de color amarillo

\* Parametros: string :: nombre\_archivo

\* Retorna: void

\*/

**bool comprobarLectura( string nombredelarchivo);**

/\* Verifica si el archivo existe en caso de que lo haga retorna true y si no lo hace retorna false

\* Parametros: string :: nombredelarchivo

\* Retorna: bool:: true or false

\*/

**void CambiarNombreArchivo(string& nombre\_archivo);**

/\* Reemplaza el string que recibe por referencia por el nuevo que le ingresaron.

\* En este caso es el nombre de un arhivo por lo que le agrega el .txt

\* Parametros: &string :: nombre\_archivo

\* Retorna: void

\*/

**bool RespuestaValida(string respuesta);**

/\* Verifica que el string que recibe sea de tamaño 1 e igual a la n o a la s.

\* En caso de serlo retorna true, si no lo es retorna false

\* Parametros: &string:: respuesta

\* Retorna: bool :: true o false

\*/

**bool intro(string& othello);**

/\* Llama a una función para verificar que esté el archivo en caso de no estarlo despliega

\* un menu que le pregunta al usario si desea salirse del programa, si no quiere salirse

\* le da la opción de ingresar otro nombre para buscar ese archivo. En caso de que sí encuentre el

\* archivo llama a una función para que imprima el nombre. Si desea seguir en el programa

\* retorna true en caso contrario retorna false.

\* Parametros: string& :: othello

\* Retorna: bool :: bandera\_titulo

\*/

**bool menu();**

/\* Llama a una función que imprime la presentación y en caso de poder hacerlo despliega el

\* menu principal del juego, en caso de que quiera jugar retorna true, si quiere salir del sistema

\* retorna false

\* Parametros: void

\* Retorna: bool :: true o false

\*/

**string SacarFechayHora();**

/\*En el momento en que se invoque la función ella invoca la función time para obtener el timepo

\* en ese momento en segundos(es el tiempo transcurrido desde el Epoch (1 de enero de 1970)

\* hasta ese momento), despupes invoca a la fucnión localtime para que convierta esa información

\* en componentes de tiempo. Para guadar la información agrupo la información como un string

\* que finalmente es lo que retorna la función

\* Parametros: void

\* Retorna: string :: fechayhora

\*/

**void CrearArchivoHistorial();**

/\* Esta función llama a tra función para que modifique un string que es el nombre del archivo

\* donde se guardan las partidas, abre el archivo en caso de que exista y si no existe lo crea

\* le coloca el título ("OTHELLO") y finalmente lo cierra

\* Parametro: string& :: nombre\_archivo

\* Retorna: void

\*/

**bool VerificarArchivo(string &nombreArchivo);**

/\* Verifica que un archivo exista, en caso de que no exista le pregunta al usuario si quiere

\* salir del programa, si no quiere hacerlo crea el archivo

\* Parametro: string& :: nombreArchivo

\* Retorna: bool :: bandera\_titulo

\*/

**void GuardarPartida(string nombre\_jugador1, int num\_fichas\_1);**

/\* Esta función llama a otra para verificar si se sencuentra el archivo, en caso de que lo haga

\* Guardo el nombre del jugador que ganó con la cantidad de fichas, la fecha y la hora

\* Parametros: string :: nombre\_jugador, int :: num\_fichas

\* Retorna: void

\*/

**void GuardarPartida(int num\_fichas);**

/\* Esta es una versión sobrecargada de GuardarPartida, la diferencia es que esta función solo

\* recibe un número entero porque se llama cuando ha ocurrido un empate

\* Parametro: int :: num\_fichas

\* Retorna: void

\*/

***Para la funcionalidad:***

**bool esNumero(string str);**

/\* Recibe un string y verifica que este sea un número entre 1 y 8 versión char

\* Parametro: string :: str

\* Retorna: bool :: true o false

\*/

**int RecibirRespuesta(string dato);**

/\* Esta función verifica que ingresen una letra entre la A y la H mayusculas

\* Parametro: string :: str

\* Retorna: bool :: true o false

\*/

**bool letraValida(string str);**

/\* Recibe un string que definirá si le pide una fila o una columna, dependiendo del caso

\* lo imprime el mensaje de cierto color y retorna el valor numérico que le corresponde al

\* tablero desde el 1 al 8

\* Parametro: string :: dato

\* Retorna: int :: numero

\*/

**bool juego();**

/\* Esta función se encarga de la continuidad del juego, instancia la clase tablero

\* y jugador, intercambia los turnos de la partida, verifica si es posible hacer jugadas y cuando

\* el juego termina invoca a la función Resultado, por último pregunta si quiere seguir jugando

\* en caso de hacerlo retorna true, en caso contrario retorna false

\* Parametro: void

\* Retorna: bool :: true o false

\*/

***Para el jugador***

**void jugador::setFicha(char nueva\_ficha)**

/\* Esta función modica el atributo ficha con el caracter que reciba

\* Parametro: char :: nueva\_ficha

\* Retorna: void

\*/

**char jugador::getFicha() const**

/\* Esta función retorna el valor del atributo ficha

\* Parametro: void

\* Retorna: char :: ficha

\*/

**string jugador::getNombre() const**

/\* Esta función retorna el valor del atributo nombre

\* Parametro: void

\* Retorna: char :: ficha

\*/

**jugador::jugador()**

/\* El constructor al momento de invocarse le solicita al usuario que le de

\* un valor al atributo nombre

\* Parametro: void

\* Retorna: void

\*/

***Para el tablero***

**tablero::tablero()**

/\*Este es el constructor de los objetos de la clase Tablero, que inicializa

\*la matriz dinámica con espacios en blanco y las cuatro fichas centrales iniciales

Parámetros: none

Retorno: void

\*/

**void tablero::imprimirTablero()**

/\* Esta función recorre los valores que hay en la matriz dentro de unos margenes prestablecidos

\* Parametro: void

\* Retorna: void

\*/

**bool tablero::MovimientoValido(int x, int y, char ficha)**

/\*Esta función se encarga de evaluar si la ficha ingresada en la posición indicada

es un movimiento válido en al menos una de todas las direcciones posibles, y guarda

valores booleanos en un arreglo que indica si alguna de esas direcciones es válida.En caso

de que la jugadano posea ninguna dirección posible, la función retorna false,pero si

tiene alguna posible, retorna true, y el arreglo de los booleanos se altera internamente

Parámetros: int :: x, int :: y, char :: ficha

Retornos: bool movimientovalido

\*/

**bool tablero::PosibleMovimiento(char ficha)**

/\* Esta función que verifica la ficha del parámetro tenga al menos un movimiento válido en el tablero

\* Parametro: char :: ficha

\* Retorna: bool :: posibilidad

\*/

**int tablero::GetPuntacion( int indice)**

/\* Esta función retorna una posición que soliciten del arreglo puntación que es un atributo de la

\* clase tablero

\* Parametro: int :: indice

\* Retorna: int :: puntacion[indice]

\*/

**bool tablero::CasillaVacia(int i, int j)**

/\* Esta función verifica si la casilla en los indices que solicitaron de la matriz está vacía

\* Parametros: int :: i, j

\* Retorna: bool :: true o false

\*/

**void tablero::ActualizarTablero(int x, int y, char ficha)**

/\*Esta función se encarga de cambiar las fichas en las direcciones que la función MovimientoValido

ha verificado válidas, accediendo internamente al arreglo de booleanos para determinar hacia dónde

cambiar las fichas

Parámetros: int :: x, y; char :: ficha

Retorno: ninguno

\*/

**bool tablero::EstadoTablero()**

/\* Esta función verifica que haya al menos un espacio en blanco en la matriz, en caso de

\* hacerlo retorna true, en caso contrario retorna false

\* Parametro: void

\* Retorna: bool :: true o false

\*/

**bool tablero::Movimiento(char ficha)**

/\* Esta función verifica que casilla en las coordenadas que ingresaron tenga movimientos válidos

\* en caso de hacerlo cuenta las fichas y actualiza el tablero e imprime la matriz,

\* si no es válido pide una nueva coordenada

\* Parametro: char :: ficha

\* Retorna: bool :: true o false

\*/

**void tablero::ContarFichas()**

/\* Recorre la matriz y cuenta cuantos caractes de las X's y las O's hay en ella,

\* luego actualiza los valores del arreglo puntacion que es un atributo de la clase tablero

\* Parametro: void

\* Retorna: void

\*/

**void tablero::Resultado(string jugadorX, string jugadorO)**

/\* Esta función imprime el nombre del jugador que ganó en pantalla y la puntación de la partida

\* Parametro: string :: jugadorX, jugadorO

\* Retorna: void

\*/

**void tablero::actualizarDirecciones()**

/\* Coloca todos los valores del arreglo direccionValida como false, este arreglo es un

\* atributo de la clase

\* Parametro: void

\* Retorna: void

\*/

**tablero::~tablero()**

/\* Además de cumplir la función normal del destructor libera la memoria dinámica que se reservó

\* para la matriz

\* Parametro: void

\* Retorna: void

\*/

1. **Implementación de algoritmos: (enlace a github)**

[**https://github.com/LsBhrqz/Parcial\_2-Informatica2.git**](https://github.com/LsBhrqz/Parcial_2-Informatica2.git)

1. **Observaciones del desarrollo y conclusiones:**
2. Es muy distinto trabajar matrices que estén guardadas en la memoria dinámica a trabajar matrices guardadas en la memoria estática, por cómo reacciona el compilador al intentar acceder a alguna posición, por lo que tratar con matrices dinámicas requiere un cuidado especial al seleccionar los índices a los que va a intentar acceder la matriz.
3. El compilador es sensible a las entradas por consola, resultando en comportamientos impredecibles cuando, por ejemplo, se espera recibir una variable tipo entero y se recibe una variable tipo char o tipo string. Se debe tener especial cuidado con cómo procesar las excepciones para la validación de entrada de datos.
4. Para los objetos, se debe tener en cuenta que sus respectivos destructores, a pesar de que pueden ser sobrecargados, no deben llamarse bajo ningún motivo en tiempo de ejecución. Esto porque el destructor es siempre automático, y al destruirlo por nuestra cuenta, en cuanto termine el ámbito para que muera el objeto, el destructor va a buscar el objeto para destruir, y al no encontrar nada, el programa se comportará de manera indefinida.
5. Es mucho más el trabajo que se debe hacer “tras las cortinas” para lograr que un programa funcione correctamente y con una estética correcta, y la experiencia que ofrece el error en este tipo de ejercicios y todos los errores que arrojan los programas en tiempo de ejecución son un nutriente esencial para nuestro proceso de análisis de futuros posibles errores que puedan presentar los cualesquiera programas que estemos realizando, así como los elementos y validaciones adicionales que hay que tener en cuenta en el desarrollo de futuros proyectos.