Samuel Di Nicola

Juan David Cárdenas

Link Repositorio: <https://github.com/juand-cardenas/examen_2_por_la_libertad/tree/master>

Link video: <https://youtu.be/wSCMPWZhLdc>

Para el segundo parcial, se debe recrear el juego llamado eothello, el cual es similar a un reversi, el cual es un juego entre 2 jugadores en el cual cada jugador tiene una ficha de un respectivo color, ya sea blanco (\*) o negro (-).

El cual se inicia con 4 fichas centrales (2 de cada color). La mecánica del juego consiste en que los jugadores deben encerrar entre 2 fichas del mismo color las fichas del color del adversario haciendo así que las fichas del contrario cambien al color de las fichas que las encierran.

El objetivo final del juego es tener la mayor cantidad de fichas del color propio posibles hasta que se llene el tablero o no haya más movimientos validos a realizar.

El eothello, se juega en un tablero de 8 por 8 como estándar, pero tiene variaciones en 6 por 6 y 10 por 10, por ende, al diseñar el juego se debe tener en cuenta esta generalización, además de las reglas propias del juego.

Para el diseño del juego se debe tener en cuenta lo siguiente:

* El tamaño del tablero
* Si la posición es valida
* Si hay movimientos disponibles
* El girar las fichas requeridas

Así de forma “gruesa” se puede describir el juego de eothello, pero para la implementación de eso en código se debe pensar en un nivel más pequeño. Ya que de esa forma se puede construir el “esqueleto” del juego en general, ya que el cuerpo se implementa aparte.

El “esqueleto” seria los métodos de la clase y el cuerpo estaría el main (el cual se creará más adelante), para este problema solo se definió una clase que sería el “juego”, el cual cuenta con sus componentes (tablero, fichas), sus reglas y comportamiento.

Donde el tablero y fichas serian atributos fijos en la clase y su comportamiento y reglas serian métodos de la clase.

Basándonos en lo anterior hablare de lo que tenemos definido en este momento:

1. Comprobar si hay movimientos
2. Capturar la posición de la ficha
3. Comprobar si la posición es valida
4. Colocar la ficha
5. Voltear las fichas encerradas
6. Imprimir el tablero
7. Contar las fichas que hay en el tablero
8. Terminar juego

Comprobar si hay movimientos: Este método comprueba si el jugador (1 o 2) tiene jugadas, si este tiene jugadas se le permitirá acceder a los otros métodos, si no tiene jugadas se pasará al siguiente turno, donde se comprobará nuevamente si tiene este tiene movimiento, si ambos jugadores no tienen jugada, se invocará terminar juego.

Capturar posición: Este método hace lo que tiene por nombre, toma la posición ingresada por el jugador en la terminal, además de ver si la posición es válida, ya que no se pude descartar la posibilidad de que ingresé valores más grandes que el tablero.

Comprobar si la posición es válida: Como en el anterior este puede asegurar que el número ingresado es válido, pero puede que ingrese una posición ya ocupada por otra ficha, además puede que la posición no cumpla con las reglas, este método se encarga de comprobar todas estas posibilidades.

Colocar ficha: Este método coloca la ficha en la posición dada por el usuario.

Voltear fichas encerradas: Cada vez que se coloca una ficha “aliada” al menos una enemiga debe ser “volteada”, este método se encarga de voltear las “enemigas” que cumple con las condiciones para ser volteada.

Imprimir tablero: se encarga de imprimir el tablero por consola cada vez que es invocado.

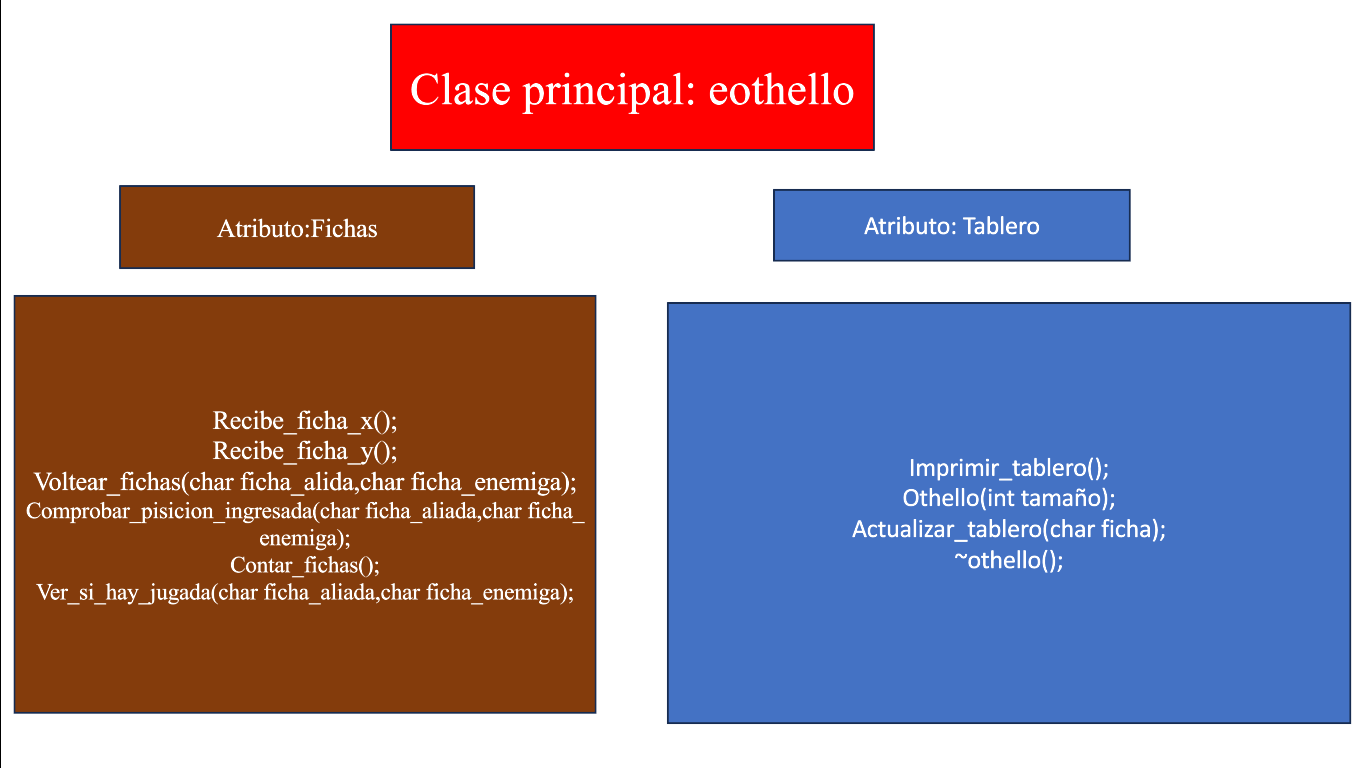
Contar las fichas que hay en el tablero: Cuando se establezca el final del juego, mediante este método se establecerá el ganador por la mayor cantidad de fichas.

Terminar juego: Este terminara el juego solo si la suma de ambos es igual a la cantidad de cuadros posibles o si ambos jugadores no tienen posibles jugadas.

Estos sería los métodos necesarios para el funcionamiento, puede que se creen nuevos métodos para realizar los principales.

Respectos a los atributos, no hay definidos, ya que estos funcionaran para guardar los datos necesarios para el funcionamiento del juego.

El main: En este lugar se llamarán los métodos y atributos, de forma que el juego se ejecute como se debe siguiendo las reglas.

Como se dijo al inicio solo se usó una clase, haciendo que el othello.cpp contuviera todos los métodos necesarios para el funcionamiento del juego.

A continuación, explicare los métodos usados en cada atributo:

Atributos: Fichas.

Void othello::recibe\_ficha\_x (); se recibe la posición ingresada en la terminal.

Este método contiene otros para su funcionamiento, los cuales son:

imprimir\_fut\_pos\_x (); imprime un texto en la terminal el cual le indica al usuario que debe ingresar la posición x en el tablero.

verificar (int x); recibe la posición ingresada por el usuario y comprueba que el número ingresado no este por fuera del tamaño del tablero, si esto no se cumple no se guardara la posición ingresada en el atributo pos\_x.

Void othello::recibe\_ficha\_y (); se recibe la posición ingresada en la terminal.

Este método contiene otros para su funcionamiento, los cuales son:

imprimir\_fut\_pos\_y (); imprime un texto en la terminal el cual le indica al usuario que debe ingresar la posición x en el tablero.

verificar\_2(int y); recibe la posición ingresada por el usuario y comprueba que el número ingresado no este por fuera del tamaño del tablero, si esto no se cumple no se guardara la posición ingresada en el atributo pos\_y.

void othello::imprimir\_error\_en\_posicion (); Este método imprime en la consola un mensaje que le indica al usuario que el número ingresado no es válido, además este método es llamado en ambos verificar

Void othello::voltear\_fichas (char fichas\_aliadas, char ficha enemiga); Este método se encarga de cambiar el color de las fichas según las reglas.

Para el funcionamiento del método hay un total de 16 for, donde ocho de estos es para determinar si hay una ficha enemiga en algunas de las posiciones cercanas a las fichas, después se comprueba si hay una ficha aliada después de las fichas enemigas. Además, se cuanta cuantos “espacios” hay entre la ficha colocada y la ficha aliada que se encuentra de forma continua a la ficha enemiga.

Todos los ocho for, hacen lo mismo, pero en direcciones diferentes, las cuales son: vertical arriba, vertical abajo, horizontal derecha, horizontal izquierdo, diagonal superior izquierda, diagonal superior derecha, diagonal inferior derecha y diagonal inferior izquierda.

Después de cada for hay otro, este solo se ejecuta si se cumple las condiciones de las fichas que explique anteriormente. Si se cumple las condiciones este cambiará las fichas dependiendo en qué dirección se halla cumplido las condiciones, si no se cumple la condición el for no se ejecutará.

Además, como se ve este recibe unos char, por ende, con solo un método es posible hacer las comprobaciones para ambas fichas.

Bool othello :: comprobar\_pisicion\_ingresada(char fichas\_aliadas, char fichas\_enemiga) Este método funciona de forma igual al anterior, solo se diferencia en que solo hay ocho fors, ya que es necesario comprobar que se cumpla la condición: al momento de poner una ficha en el tablero se debe voltear al menos una ficha enemiga al color “aliado”, por ende se debe comprobar las ocho direcciones que rodean la posición ingresada, otra diferencia es que en lugar donde se encuentra los fors para cambiar el “color” de las fichas se encontrara un return true, sino se cumple la condición en las ocho direcciones habrá un return false.

Si sale este último, no se actualizará el tablero ni se volteará las fichas, además de imprimir en consola dicha situación, lo anterior esta echo en el main.cpp, no en el propio método.

Además, como se ve este recibe unos char, por ende, con solo un método es posible hacer las comprobaciones para ambos turnos.

Bool othello::ver\_si\_hay\_jugada(char ficha\_aliada,char ficha\_enemiga); El método es usado para comprobar si el jugador tiene turno, ya que puede haber la posibilidad de que si se entrega el turno este no pueda poner ficha ya que no cumple con las reglas, por ende el método evalúa el tablero hasta hallar una posición donde el jugador de turno pueda poner una ficha, si no puede evaluar al otro jugador, si ninguno tiene la posibilidad acabara con el juego, así fue establecido en el mai.cpp.

El método está construido de igual forma que el anterior método descrito, solo se diferencia en que este método evalúa varias posiciones en el tablero, en lugar de evaluar una posición fija. Al momento de hallar una posición se regresará un true.

Los anteriores métodos, fueron construidos para que se comportaran como “árbitros” quienes se dedican a revisar que el juego se lleve a cabo de forma correcta, por esa razón se construyeron los métodos anteriormente descrito.

Atributos: tablero.

Void othello::imprimir\_tablero (); La función recorre la matriz de inicio a fin, mientras se recorre se imprimirá en la consola lo contenido en la posición, de esa forma se construirá el tablero, además de haber cout «extras para la estética del tablero.

Othello::othello (int tamaño); este es el constructor donde se creará el tablero que se usará para el juego, el tamaño del tablero esta “establecido”, pero puede ser cambiado, para usar sus otras versiones.

Void Othello::actualizar\_tablero (char ficha); El método, pondrá la ficha que este estipulada, en la posición ingresada por el usuario, esto solo se realizara si es posible poner y cambiar las fichas, para así evitar errores.

Othello::~othello (); Este es el destructor de la clase, el cual borrara el tablero cuando se acabe el juego.

Estos son los métodos necesarios para el desarrollo adecuado del juego, ya que el tablero es una parte fundamental del juego, por eso se desarrolló los métodos descritos.

Evolución de la solución y consideraciones de la implementación:

El principal problema que afrontamos fue a mediados del desarrollo en donde teníamos problemas respecto a las validaciones relacionadas con la posición, por el hecho de que había que comprobar todas las posibilidades antes de dar una respuesta final, haciendo así que fuera complicado conectar las validaciones pertinentes. Resumiendo, el caso es que había que validar 3 cosas: Si la posición estaba o no ocupada, si había una ficha aliada en cualquiera de la dirección en donde se iba a colocar la ficha y si entre esas 2 fichas aliadas había fichas enemigas las cuales permitían el juego. Luego de discutirlo y darle varias vueltas, decidimos reunir todo en una misma función en donde dependiendo de las circunstancias, una variable iba a tomar valor de true (La cual mantenía el ciclo establecido en el mail, haciendo referencia a que se debía volver a jugar porque la posición no era válida), o false (La cual daba pie a romper el ciclo delimitador y continuando el transcurso normal del juego.