

**INFORMATICA II**

**PARCIAL II**

**Elaborado por:**

**MESA ROLDAN CARLOS ANDRÉS**

**MESA ROLDAN JOSÉ DAVID**

**Presentado a:**

**GUERRA SOLER ANIBAL JOSE**

**INGENIERÍA TELECOMUNICACIONES**

**FACULTAD DE INGENIERÍA**

**UNIVERSIDAD DE ANTIOQUIA**

**2023**

1. Contextualización

Othello, también conocido como Reversi, es un juego de estrategia abstracta que se juega en un tablero cuadrado de 8x8. Aunque su origen exacto es incierto, se cree que se inspiró en un juego antiguo llamado "Reversi" que fue popularizado en Inglaterra a finales del siglo XIX. Sin embargo, Othello tal como lo conocemos hoy en día fue creado por el japonés Goro Hasegawa en 1971.

El juego utiliza fichas de dos colores, generalmente negro y blanco. Los jugadores se turnan para colocar fichas en el tablero, con el objetivo de rodear las fichas del oponente y convertirlas en sus propias fichas. Una ficha se coloca en un espacio vacío del tablero, adyacente a una ficha del oponente (en cualquier dirección, se de manera diagonal, vertical u horizontal), de tal manera que al menos una ficha del oponente quede atrapada entre la nueva ficha y otra ficha del jugador que coloca. Todas las fichas del oponente que quedan atrapadas de esta manera se voltean y se convierten en fichas del jugador que colocó la última ficha.

El juego se gana cuando el tablero está lleno y el jugador con más fichas de su color en el tablero gana la partida, sin embargo, se puede dar el caso de que uno de los jugadores capture todas las piezas del rival, generando la victoria para quién aun conserva sus fichas en el tablero.

Othello ha sido ampliamente jugado a nivel mundial y ha inspirado numerosos torneos y competiciones, ha sido implementado con distintas variaciones, entre las que se destaca la minimización o maximización del tablero, entro otras tantas variables.

1. Análisis

Como se puede observar anteriormente, se cuenta con una serie de retos para la parametrización y representación del juego otello. A continuación, se enlistarán una seria de retos a ser resueltos en el transcurso de la implementación del juego:

* Estructura de datos apropiada, para la representación del tablero (Sea dinámica o estática y justificar el por qué).

Se eligen estructuras de memoria estática, dado se hará uso de ellas durante todo el tiempo de ejecución del programa. Básicamente se tendrá una matriz, la cual será modificada periódicamente, dado que los jugadores estarán constantemente modificando las posiciones de sus fichas. Y otra matriz que será de ayuda para identificar los movimientos posibles, esta matriz será de tipo bool.

* Lógica y condiciones que permitan validar y verificar los posibles movimientos de cualquier jugador. Basándonos en el concepto de los vectores unitarios y teniendo la posición que se desea evaluar, se proceden a explorar las distintas orientaciones, izquierda, derecha, diagonal superior derecha, diagonal inferior derecha, etc. Donde el objetivo será encontrar un rival o más y posteriormente una posición vacía, para crear el llamado sándwich.
* Implementación de métodos apropiados para identificar el final del juego, sea porque un jugador se retira, porque el tablero se ha llenado o simplemente porque un jugador ha perdido todas sus fichas. Se tienen dos condiciones para terminar el juego, una por mutuo acuerdo, donde se da una combinación de teclas para terminar el juego. Otra será dada por una función que evaluará si el arreglo se ha llenado completamente.
* Representación amena de un tablero de Otelo, el cual, si bien se da en la consola, puede ser agradable y ameno. Se utilizo la librería window, con el objetivo de inyectar colores, para crear un ambiente ameno para los jugadores.
* Implementación de un historial en un documento .txt, el cual permita visualizar los nombres de los jugadores, su puntaje y la fecha en la que se dio origen a la partida.

1. Diseño.

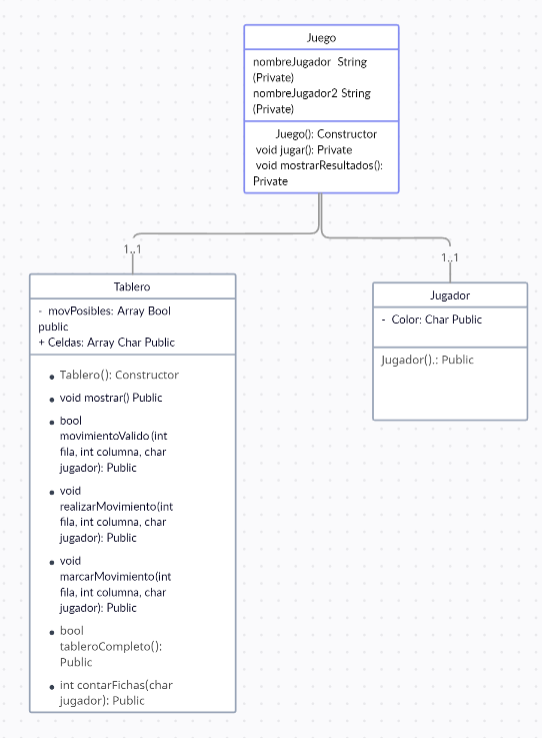


Ilustración 1: Diseño del juego

En la ***ilustración 1*** se pueden apreciar las distintas clases, con sus respectivos atributos y métodos. La clase juego es la única que utiliza métodos y atributos privados, debido a que en su método jugar se desarrollará prácticamente todo el juego. Tablero y Jugador al presentar públicamente todos sus integrantes (métodos y atributos) alimentan el contenido de la clase juego, la cual, después de la clase main, es la principal y la más importante.

En la clase tablero se presentan distintos métodos, los cuales por su nombre dan a entender fácilmente su papel en el juego. Sin embargo, hay dos métodos muy importantes que influyen directamente en las dinámicas del juego, estos son: marcarMovimiento y movimientoValido. Dichos algoritmos tienen como base el funcionamiento y las dinámicas de los vectores unitarios, los cuales tienen una dirección y procederán a profundizar en dicha dirección para determinar qué tan viable es colocar la ficha en cierta posición, a través de validaciones que se ajustan a las reglas de Otello.

En la clase juego se abordará un ciclo, el cual se muestra como la base de clase, dicho método tendrá una durabilidad dada por un ciclo, el cual otorgará constantemente el turno a cada jugar. Cada vez que realice un movimiento valido, se mostrar el arreglo dimensional del estado del juego, por ello era importante utilizar memoria estática, la cual es útil durante todo el tiempo de ejecución del juego.

A continuación, se procederá a detallar en los principales métodos del juego con sus respectivas clases.

**Clase Tablero:**

**Constructor Tablero::Tablero()**

Este constructor inicializa un tablero de tamaño TAMANO (definido en el archivo "Tablero.h") con espacios en blanco y coloca las fichas iniciales de forma específica: dos fichas de cada tipo ('\*' y '-') en el centro del tablero.

**Método void Tablero::mostrar() const**

Este método muestra el estado actual del tablero en la consola. Utiliza colores para representar las fichas de cada jugador: rojo para '\*', verde para '-', y azul para los puntos de movimiento válidos. Además, imprime el número de fila y columna para ayudar a los jugadores a entender la disposición del tablero.

**Método bool Tablero::movimientoValido(int fila, int columna, char jugador) const**

Este método verifica si un movimiento en una fila y columna específicas es válido para un jugador dado. Un movimiento es válido si la posición está dentro del tablero, es una celda vacía y existe al menos una ficha del oponente adyacente en cualquier dirección. Si se cumple esta condición, el método devuelve true; de lo contrario, devuelve false.

**Método void Tablero::marcarMovimientosPosibles(char jugador)**

Este método marca en el tablero las posiciones donde el jugador actual puede realizar movimientos válidos. Utiliza el método movimientoValido para determinar estas posiciones y las almacena en el arreglo movimientosPosibles.

**Método void Tablero::realizarMovimiento(int fila, int columna, char jugador)**

Este método coloca una ficha del jugador en la posición especificada y captura las fichas del oponente según las reglas del juego Othello. El método busca en las ocho direcciones posibles desde la posición dada y captura las fichas del oponente si están atrapadas entre dos fichas del jugador actual.

**Método bool Tablero::tableroCompleto() const**

Este método verifica si el tablero está completo, es decir, si todas las celdas están ocupadas por fichas (ya sea del jugador '\*' o '-'). Si el tablero está completo, devuelve true; de lo contrario, devuelve false.

**Método int Tablero::contarFichas(char jugador) const**

Este método cuenta y devuelve el número de fichas que el jugador tiene en el tablero.

**En resumen, esta clase Tablero se encarga de mantener el estado del tablero del juego Othello, determinar los movimientos válidos, realizar los movimientos de los jugadores, verificar si el tablero está completo y contar el número de fichas de cada jugador.**

**Clase Juego:** La clase Juego controla el flujo del juego, permite a los jugadores realizar movimientos, determina el ganador y guarda los resultados del juego en un archivo.

**Constructor Juego::Juego(const string& nombre1, const string& nombre2)**

Este constructor toma dos nombres de jugadores como argumentos y los almacena en nombreJugador1 y nombreJugador2. También inicializa los símbolos de los jugadores ('\*' para el jugador 1 y '-' para el jugador 2).

**Método void Juego::jugar()**

Este método maneja la lógica principal del juego. Permite a los jugadores realizar movimientos alternados, mostrando el tablero y solicitando las coordenadas del movimiento a los jugadores. El juego continúa hasta que el tablero está completo o ambos jugadores deciden pasar su turno. Al final del juego, muestra el ganador y guarda los resultados en un archivo llamado "resultados.txt".

**Método void Juego::mostrarResultados()**

Este método muestra los resultados almacenados en el archivo "resultados.txt". Lee el archivo línea por línea y muestra su contenido en la consola.

**Método void Juego::guardarResultado(const std::string& ganador, int contadorNegras, int contadorBlancas)**

Este método guarda los resultados del juego en el archivo "resultados.txt". Añade la fecha y hora actual, los nombres de los jugadores, el ganador y la puntuación de cada jugador al final del archivo.

1. <https://github.com/cmesaUdea/OthelloGame/tree/carlos>
2. Experiencia de aprendizaje.

* Se tuvieron bastantes problemas con la primera implementación, dado que se deseaba utilizar totalmente memoria dinámica, pero esta no fue posible, dado que era necesario guardar durante el tiempo de ejecución del problema la información presente en el arreglo de las celdas. No era posible darse el lujo de liberar la información en algún momento diferente al final del juego.
* Se hizo un uso apropiado de lo viste en el modulo de tratamiento de archivos de texto. La clase Juego muestra cómo leer y escribir archivos para almacenar y recuperar datos (en este caso, resultados del juego).
* La clase Juego demuestra un control de flujo efectivo utilizando bucles y condiciones para gestionar el flujo del juego, alternando turnos entre jugadores y verificando la validez de los movimientos.
* Las clases representan abstracciones del mundo real (un tablero de juego y un juego de Othello), mostrando cómo los objetos en el programa se relacionan y se comportan.
* La implementación de la clase Jugador facilito el desarrollo del juego, dado que se tuvieron problemas para intercalar entre jugadores de manera efectiva.