CONTEXTUALIZACION:

Reversi es un juego de mesa que se juega sobre una grilla, de modo que tendremos que usar un sistema de coordenadas Cartesiano con coordenadas XY. Es un juego para dos jugadores.

Reversi tiene un tablero de 8 x 8 y baldosas que son negras de un lado y blancas del otro (nuestro juego las reemplazará por "*" y "o" y además se dejará el tamaño del tablero abierto para una matriz cuadrada de x tamaño de fila al igual de columnas). Este tendrá un tablero inicial el cual se ilustra en la siguiente figura. El jugador negro y el jugador blanco toman turnos para colocar una nueva baldosa de su color. Cualquier baldosa del oponente que se encuentre entre la nueva baldosa y las otras baldosas de ese color es convertida. El objetivo del juego es tener tantas baldosas de tu color como sea posible.

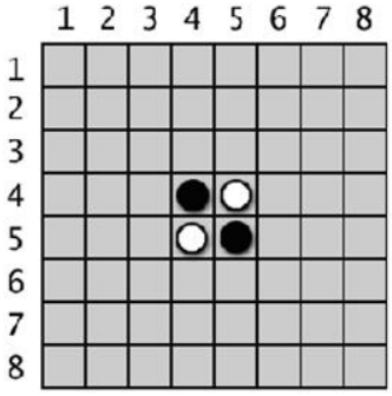


Figura 15-1: El tablero inicial en Reversi tiene dos baldosas blancas y dos negras.

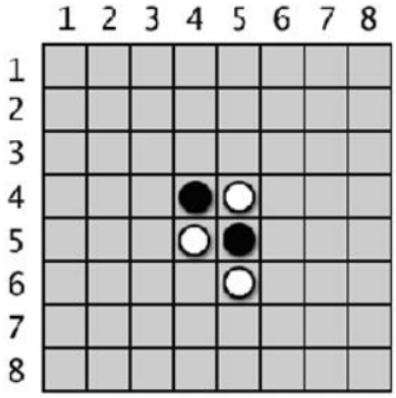


Figura 15-2: El jugador blanco coloca una nueva baldosa.

A continuación, se pueden observar algunos ejemplos de movimientos.

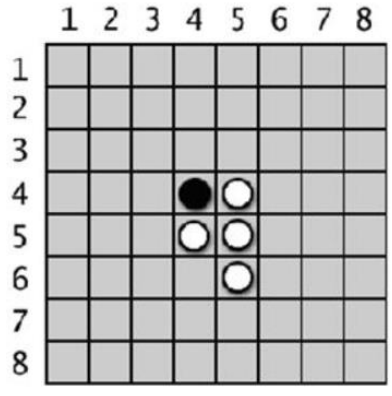


Figura 15-3: La movida del jugador blanco convierte una de las baldosas negras.

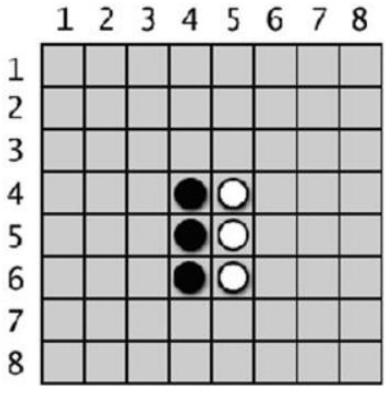


Figura 15-4: El jugador negro coloca una nueva baldosa, la cual convierte una de las baldosas blancas.

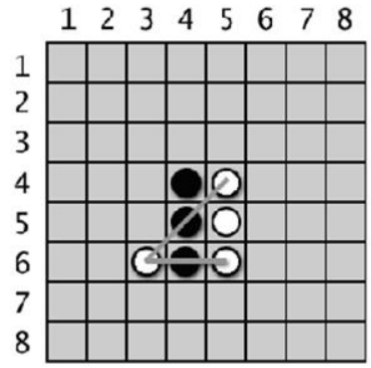


Figura 15-5: La segunda movida del jugador blanco en 3, 6 convertirá dos baldosas negras.

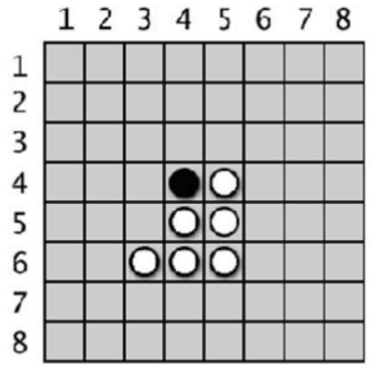


Figura 15-6: El tablero luego de la segunda movida del jugador blanco.

Como condicional del movimiento para los jugadores tenemos que deben hacer siempre jugadas que capturen al menos una baldosa y además el juego terminará cuando ningún jugador puede seguir moviendo, o el tablero está completamente lleno. Gana el jugador con más baldosas de su color.

ANALISIS:

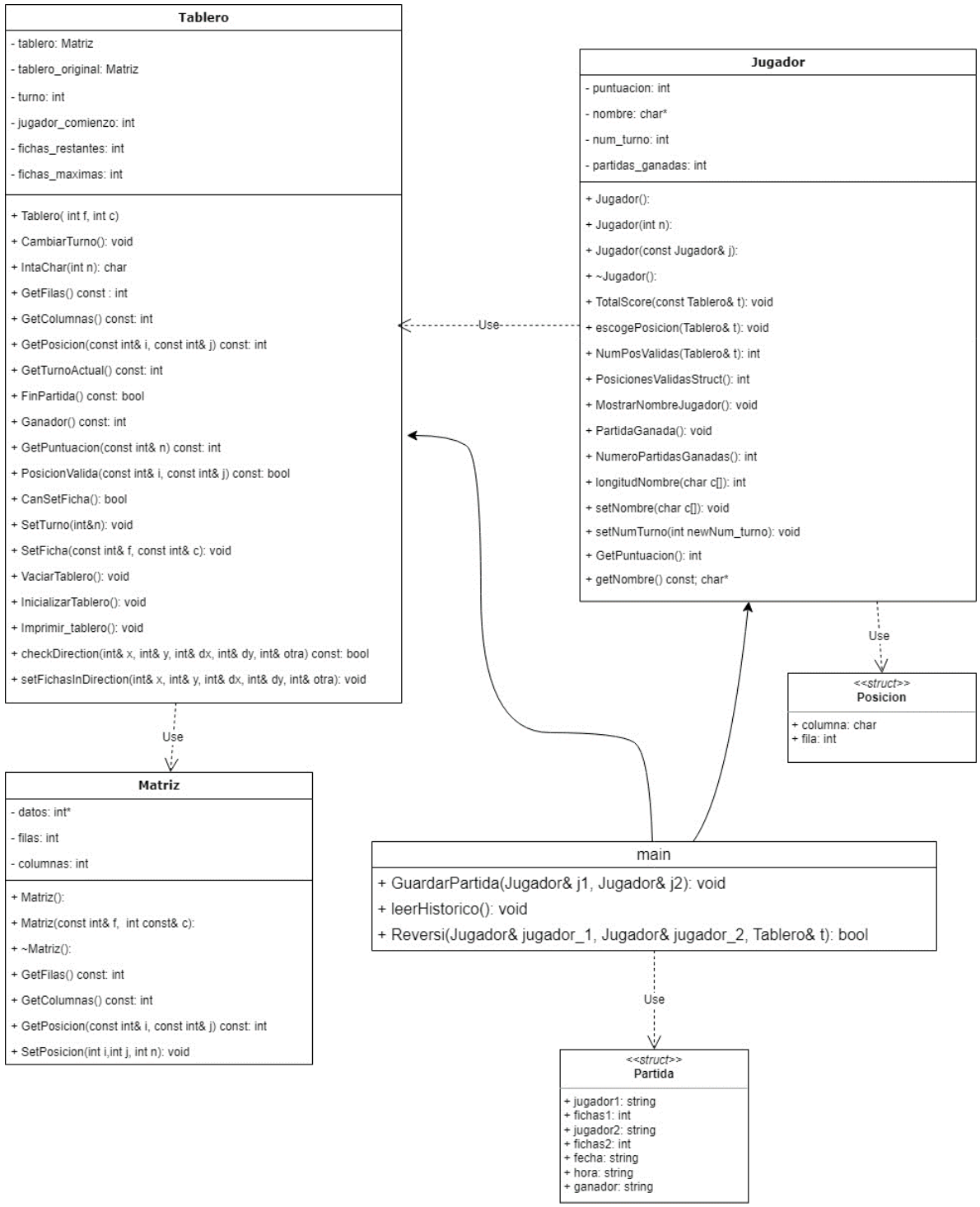
consideraciones para el abordaje del problema y desarrollo de propuesta de la estrategia de solución.

consideraciones importantes para el abordaje de este problema:

1. **Dejar el tamaño del tablero abierto**
2. **Será de dos jugadores**
3. **Hay que tener en cuenta el modelamiento de objetos, para darle vida al tablero y al jugador**
4. **El modo de mover la ficha, y el condicional de que si o si tiene que haber una ficha del rival entre dos del jugador contrario.**
5. **El archivo que guardará el historial de las partidas.**

Se propusieron 3 clases para resolver este problema: Jugador, Tablero, Matriz.

Diagrama de clases:


Tablero
- tablero: Matriz
- tablero_original: Matriz
- turno: int
- jugador_comienzo: int
- fichas_restantes: int
- fichas_maximas: int
+ Tablero(int f, int c)
+ CambiarTurno(): void
+ IntaChar(int n): char
+ GetFilas() const : int
+ GetColumnas() const: int
+ GetPosicion(const int& i, const int& j) const: int
+ GetTurnoActual() const: int
+ FinPartida() const: bool
+ Ganador() const: int
+ GetPuntuacion(const int& n) const: int
+ PosicionValida(const int& i, const int& j) const: bool
+ CanSetFicha(): bool
+ SetTurno(int&n): void
+ SetFicha(const int& f, const int& c): void
+ VaciarTablero(): void
+ InicializarTablero(): void
+ Imprimir_tablero(): void
+ checkDirection(int& x, int& y, int& dx, int& dy, int& otra) const: bool
+ setFichasInDirection(int& x, int& y, int& dx, int& dy, int& otra): void
Jugador
- puntuacion: int
- nombre: char*
- num_turno: int
- partidas_ganadas: int
+ Jugador():
+ Jugador(int n):
+ Jugador(const Jugador& j):
+ ~Jugador():
+ TotalScore(const Tablero& t): void
+ escogePosicion(Tablero& t): void
+ NumPosValidas(Tablero& t): int
+ PosicionesValidasStruct(): int
+ MostrarNombreJugador(): void
+ PartidaGanada(): void
+ NumeroPartidasGanadas(): int
+ longitudNombre(char c[]): int
+ setNombre(char c[]): void
+ setNumTurno(int newNum_turno): void
+ GetPuntuacion(): int
+ getNombre() const; char*
Use
Use
Use
Use
<<struct>>
Posicion
+ columna: char
+ fila: int
Matriz
- datos: int*
- filas: int
- columnas: int
+ Matriz():
+ Matriz(const int& f, int const& c):
+ ~Matriz():
+ GetFilas() const: int
+ GetColumnas() const: int
+ GetPosicion(const int& i, const int& j) const: int
+ SetPosicion(int i,int j, int n): void
main
+ GuardarPartida(Jugador& j1, Jugador& j2): void
+ leerHistorico(): void
+ Reversi(Jugador& jugador_1, Jugador& jugador_2, Tablero& t): bool
<<struct>>
Partida
+ jugador1: string
+ fichas1: int
+ jugador2: string
+ fichas2: int
+ fecha: string
+ hora: string
+ ganador: string

En la clase Tablero diseñamos todas las funciones relacionadas a las posiciones de las fichas en el tablero, haciendo uso de una matriz dinámica como objeto de la clase Matriz, decidimos crear una clase dedicada a la matriz para que el programa sea modular si en algún momento se quieren agregar nuevas funcionalidades al programa ya sea que usen la clase tablero o se necesite un objeto matriz para otra cosa.

Se creó una clase para el Jugador ya que tendremos más de 1, facilitándonos el desarrollo del programa haciendo uso de POO.

Los principales métodos de la clase matriz son:

Su constructor sobrecargado Matriz::Matriz(const int& f, const int& c): el cual se encarga de crear la matriz dinámica con un tamaño especifico.

El método void Matriz::SetPosicion(int i, int j, int n): su función es darle un valor a una posición especifica de la matriz.

En la clase Tablero tenemos como atributos un objeto de la clase matriz con el cual vamos a operar con sus principales metodos:

void Tablero::InicializarTablero() : con este método inicializamos el tablero con sus posiciones vacioas excepto las 4 del centro, en las que ponemos las fichas iniciales.

void Tablero::Imprimir_tablero() : con esta imprimimos el tablero en la consola con sus filas y columnas de manera cómoda para el usuario.

bool Tablero::PosicionValida(const int& x, const int& y) const: Verifica si una posición específica en el tablero es válida para hacer un movimiento.

bool Tablero::CanSetFicha(): Verifica si el jugador actual puede hacer un movimiento.

void Tablero::SetFicha(const int& x, const int& y): verifica con la función anterior si el jugador puede hacer un movimiento valido, si no puede le cede el turno al otro jugador, pero si tiene movimientos validos coloca una ficha del jugador actual en una posición específica y voltea las fichas del oponente en consecuencia.

void Tablero::setFichasInDirection(const int& x, const int& y, const int& dx, const int& dy, const int& otra): con esta function volteamos las fichas del oponente.

bool Tablero::checkDirection(const int& x, const int& y, const int& dx, const int& dy, const int& otra) const: Verifica si al colocar una ficha en una posición específica en una dirección específica, se voltearía al menos una ficha del oponente, si es verdadero la funcion imprimir matriz va a poner un punto en esa posición, esto con el fin de que el usuario tenga una ayuda visual de donde tiene movimientos válidos.

En la clase Jugador creamos un struct Posición con las filas y columnas como variables del struct para poder operar con algunos métodos de la clase tablero.

Los principales métodos de Jugador son:

bool Jugador::escogePosicion(Tablero& t): con esta funcion el jugador puede escoger una posición en el tablero y se hacen todas las verificaciones pertinentes acorde al tablero de juego.

int Jugador::NumPosValidas(Tablero& t) : con esta calculamos cuantas posiciones validas tiene el jugador.

Los demás métodos son los setters y getters para hacer uso de ellos en la funcion main del programa.

En el main del programa se creó una estructura struct Partida en la cual almacenaremos la información de la partida.

En el main tenemos estas funciones:

void GuardarPartida(Jugador& j1, Jugador& j2): con esta función guardamos los resultados de cada partida en un archivo de texto no sin antes verificar que si exista el archivo.

void leerHistorico(): con esta función leemos el archivo te texto en el que guardamos los resultados para imprimir el historial de las partidas.

bool Reversi(Jugador& jugador_1, Jugador& jugador_2, Tablero& t): con esta funcion controlamos el flujo del juego Reversi. En cada turno, verifica si el jugador actual puede hacer un movimiento. Si es posible, el jugador elige una posición en el tablero para colocar su ficha. Este proceso se repite hasta que el juego termina. Al final del juego, se imprime el tablero y se anuncia el ganador. También se actualizan las puntuaciones de los jugadores y se guarda la partida.

Se tiene un menú con 4 opciones para el usuario, jugar, ver historial, reglas y salir.

En la opción jugar pedimos el tamaño de la matriz y los nombres de los jugadores para así invocar la función Reversi() e iniciar el juego.

En la opción 2 invocamos la función leerHistorico() para imprimir en pantalla el historial de las partidas.

En la opción 3 imprimimos en pantalla las reglas del juego Reversi para que el usuario tenga claro cómo funciona.

Y por último salir del menú.

Enlace repositorio: https://github.com/daviddsuarezz/parcial2.git

Experiencia de aprendizaje: a lo largo del desarrollo nos encontramos con algunas dificultades a la hora de programar La lógica para chequear las posiciones validas y para hacer el cambio de las fichas encerradas, haciéndonos considerar mejor algunos métodos que estábamos implementando, así como la eliminación de otros, pero luego de poder superar estos problemas aprendimos mucho sobre el manejo de matrices de manera lógica.