Parcial 2 Informática II:

Implementación Juego de mesa Othello

Yuliana Betancur Saldarriaga

Universidad de Antioquia

Facultad de ingeniería

Informática II

Medellín – Antioquia

Juego Othello

Othello es un juego de estrategia diseñado para dos jugadores, este es ejecutado mediante el uso de un tablero de 8x8. Como objetivo principal se requiere contar con la mayoría de fichas del color de determinado jugador en el tablero al final del juego, estas fichas se capturan mediante una funcionalidad donde son rodeadas entre dos fichas del oponente. El desarrollo del juego de forma digitalizada y programada implica realizar la implementación de la lógica de juego en una interfaz de usuario donde se pueda jugar contra un oponente.

Análisis y estrategia de solución

Para poder realizar la implementación, se consideraron diferentes aspectos, entre estos se encuentran:

- Representación del tablero y las fichas: Estos deben ser representados de forma óptima y clara para el usuario, además deben realizarse movimientos que deben ser validados previamente por el programa de acuerdo a las reglas de juego establecidas donde se tiene un límite para que un movimiento se considere válido.
- Gestión de turnos: Los movimientos deben realizarse de forma alternada entre los jugadores, adicionalmente se debe tener en cuenta la validez de los movimientos disponibles ya que en caso de que no se cuente con dicha validez el usuario debe ceder su turno.
- Interacción entre jugadores: Las diferentes jugadas pueden afectar las fichas del otro jugador, esto debe verse reflejado en el tablero para la gestión posterior del usuario que sigue con el turno.
- Control de estado: Debe gestionarse el estado del juego, verificando si existen jugadas disponibles, la finalización del juego, el manejo de fichas para determinar un ganador, etc.

La estrategia de solución implicó la creación de 5 clases principales:

- **Clase Tablero**: Gestiona el tablero teniendo en cuenta inicialización, validación y ejecución de movimientos, conteo de fichas y determinación de jugador ganador.
- **Clase Jugador**: Manejo de identidades de los jugadores, identifica el nombre y ficha de cada uno.
- **Clase Juego:** Gestiona el proceso del juego, incluyendo la alternancia de turnos, manejo de inicio/finalización y captura de resultados.

- **Clase Partida:** Registra y almacena los resultados de cada partida jugada, con la fecha y hora, nombres de los jugadores y el ganador con su respectiva cantidad de fichas.
- **Clase Historial:** Almacena el histórico de partidas ejecutadas dentro del juego tomando como referencia en cada ejecución los datos capturados en la clase partida.

Diseño

Clase tablero

Atributos:
char tablero: Representa el tablero del juego.
int bsize: Usada para la representación de la matriz para definir el tamaño del tablero.
int fichasj1: Variable para llevar conteo de fichas del Jugador1.
int fichasj2: Variable para llevar conteo de fichas del Jugador2.

Métodos:
void **startb**(char ficha1, char ficha2); Inicializa el tablero con las fichas iniciales
void **imprimir**(); Imprime el estado actual del tablero
bool **movimientovalido**(int fila, int columna, char ficha); Verifica si un movimiento es válido de acuerdo a unas reglas establecidas.
bool **movimiento**(int fila, int columna, char ficha); Ingresa una ficha en el tablero si el movimiento es válido.
bool **ifmovalid**(char ficha); Verifica si un jugador tiene movimientos válidos para posteriormente definir si se debe ceder el turno.
void **giro**(int fila, int columna, char ficha); Evalúa las fichas donde se genera un encierro tipo ‘sandwich’ y realiza el cambio de acuerdo a las reglas establecidas.
void **cfichas**(char ficha1, char ficha2); Lleva el conteo de fichas por cada jugador en cada turno.
string **ganadorp**(string jug1, string jug2); Determina el jugador ganador del juego.

Clase Jugador

Atributos Privados:
string nombre: Representa el nombre del jugador.
char ficha: Representa la ficha que el jugador usará en el juego.

Métodos:
string **getn**(); Retorna el nombre del jugador.
char **getf**(); Retorna la ficha del jugador.

Clase Juego

Atributos:
Jugador* jugadores[2]: Arreglo de punteros a Jugador representando a los jugadores del juego.
Tablero* tablero: Puntero a Tablero representando el tablero del juego.
Partida* partida: Puntero a Partida para captura de información del juego.
int contador=0: Contador de jugadas, para determinar casillas disponibles.
string resultadospartida: Variable para almacenar los resultados de la partida, posteriormente consumida por la clase 'partida'

Métodos:
Juego(): Constructor de la clase.
~**Juego**(): Destructor de la clase.
void **start**(): Inicia el juego, creando los jugadores.
void **turno**(): Controla el turno de los jugadores.
void **partidaj**(): Inicia una partida, creando el tablero.
int **finpartida**(): Lleva el contador para saber cuántas casillas se encuentran disponibles.
bool **findejuego**(): Determina el fin de la partida.
void **resfinal**(): Imprime el tablero final cuando termina la partida.
void **respartida**(): Almacena el resultado de la partida para asignarlo a la clase partida.
Partida* **gpartida**(): Retorna el puntero a la partida en curso.

Clase Partida

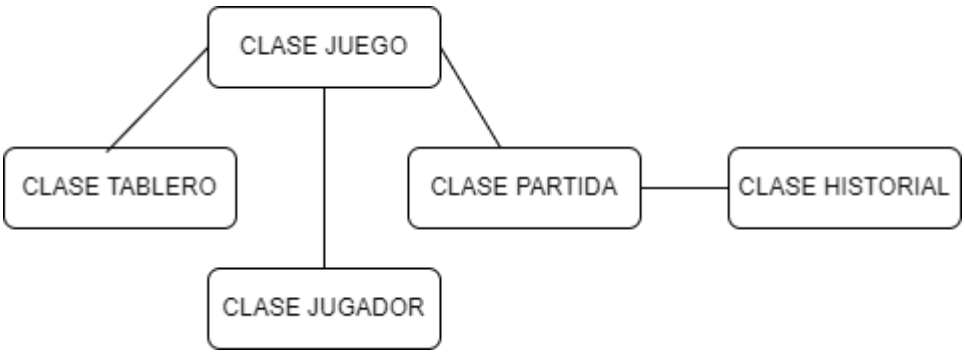
Atributos:
string fechayhora;
string resultadoPartida;
Historial historial;

Métodos:
void **initpartida**(): Retorna un mensaje de inicio de partida
string **reghorayfecha**(): Registra la hora y fecha de la partida
void **sresultadop**(string resultadoPartida): Establece el resultado de la partida
string **gresultadop**(); Retorna el resultado de la partida

Clase Historial:

Métodos:
  void **guardar**(const string& resultado): Guarda un resultado en el historial.
  void **imprimir**(): Imprime todos los resultados almacenados en el historial.

Aunque las clases gestionan procesos diferentes, se encuentran relacionadas y se usan unas a otras para generar la armonía del juego.
El diagrama de relación de clases es el siguiente:



La clase juego toma elementos y métodos de la clase tablero para generar el tablero y hacer la gestión, así mismo para llevar este control usa la clase jugador para identificar los jugadores y sus fichas correspondientes.
De igual forma la información que captura es pasada a la clase partida para el almacenamiento de los resultados finales de la partida jugada.
Finalmente, la clase historial consume la información generada en la clase partida con los resultados finales y los almacena dentro de un archivo de texto que se llena cada vez que se juegue una nueva partida.

Con este tipo de implementación se busca diferenciar los componentes para su entendimiento y mantenimiento en caso de ser necesario, practica muy común en el ambiente de programación.

Algoritmos implementados:

El repositorio que contiene los archivos fuente utilizados para la solución se encuentra en el siguiente enlace:
https://github.com/YulianaBSu/Parcial2.git

Dentro del directorio Parcial2 se alojan los archivos que contienen la declaración de las clases y la implementación de estas:
https://github.com/YulianaBSu/Parcial2/tree/master/Parcial2

Experiencia de aprendizaje:

Durante el desarrollo del proyecto fueron enfrentados diversos desafíos, entre estos:

- Gestión adecuada de clases y objetos: Esto incluye temas como la validación de cambios entre clases e interacción entre estas, donde era necesario definir y utilizar diversos miembros, métodos y constructores, además también se llevó a cabo el uso de punteros para gestionar objetos en la memoria.
- Optimización de consumo de memoria dinámica: Mediante la asignación y liberación de memoria con el uso de punteros, esto para garantizar una ejecución eficiente del programa y los recursos de la máquina.
- Captura adecuada de información generada entre diferentes clases: Haciendo uso de diversos medios como los punteros y arreglos que facilitan el uso de la información sin recurrir a la duplicación de código y aprovechando los recursos implementados.
- Implementación de condicionales: Para validaciones en el desarrollo del juego, como movimientos válidos, gestión de turnos (ejecutar y ceder), finalización del juego, manejo de fichas a lo largo de la partida, etc.
- Manejo de excepciones y errores: Haciendo uso de mecanismos que gestionaban los inconvenientes que podrían surgir a lo largo de la ejecución.

El proyecto proporcionó una amplia experiencia en el ámbito del desarrollo de software en C++, abarcando desde la parte conceptual de un problema hasta la implementación de una solución.
Fueron adquiridas habilidades esenciales en programación orientada a objetos, diseño de algoritmos y manejo de memoria, además de tener en cuenta la implementación de buenas prácticas en el campo de la programación.