Informe Desarrollo Parcial II

Juan Diego Carrera Quintero

Manuel Esteban Orjuela Montealegre

Universidad de Antioquia

Facultad de Ingeniería

Departamento de Electrónica y Telecomunicaciones

6 de noviembre de 2023

Contextualización del Problema

La programación orientada a objetos (POO) es un paradigma esencial en el desarrollo de software, ya que modela entidades del mundo real mediante la encapsulación en objetos. Esta herramienta versátil facilita la comprensión y diseño de conceptos, permitiendo la creación de software realista, por lo cual es un término fundamental en la solución propuesta de éste parcial.

El problema planteado es crear y modelar el juego estratégico de mesa Othello. La idea principal consiste en simular una partida de dos jugadores en tiempo real teniendo en cuenta las indicaciones y restricciones asignadas con el fin de crear un programa totalmente funcional.

Análisis

En un inicio, optamos por presentar nuestra solución a través de dos clases: una diseñada para gestionar de manera integral todas las mecánicas y características del juego, y otra destinada a administrar los archivos de texto que contendrán un registro completo de las partidas jugadas.

Consideraciones para la Clase Juego

Dadas las restricciones relacionadas con el uso de contenedores, hemos optado por la implementación de arreglos dinámicos para la construcción y desarrollo de cada partida. Asimismo, hemos incorporado funciones específicas, como movimientos y cambios de jugador, como métodos privados de la clase.

Consideraciones para la Clase Manejo de Archivos

En esta sección, se han tenido en cuenta aspectos relacionados con el manejo de excepciones. Hemos focalizado nuestra atención principalmente en los casos de búsqueda y apertura de archivos durante la creación de la clase, así como en la reescritura de la información.

```
Jugadores: Jugador X, Jugador O
Fecha: 5/11/2023
Hora: 14:17
Ganador: Jugador X
Fichas: 4

Jugadores: Jugador X, Jugador O
Fecha: 5/11/2023
Hora: 14:45
Ganador: Empate
Fichas: 2
```

Formato Contenido Archivo de Texto

Diseño

Clase OthelloGame

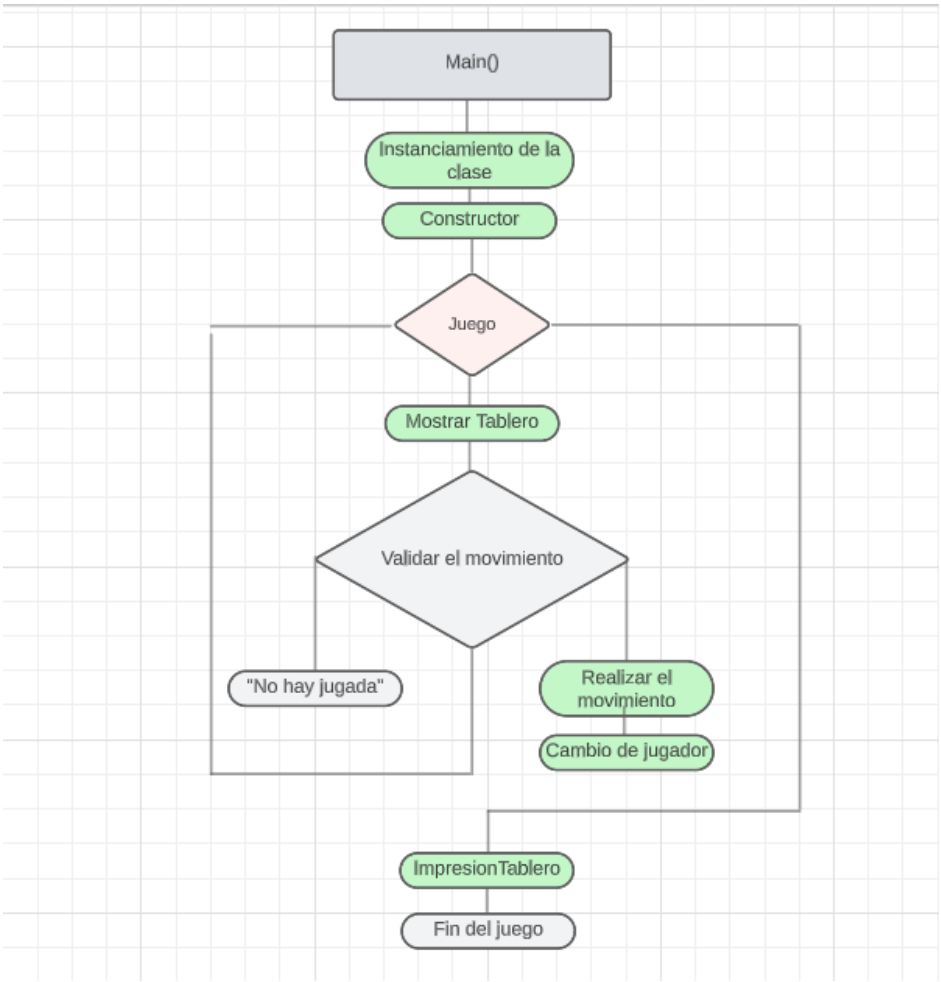


Ilustración 1. Diagrama De Flujo – Lógica De La Clase

Descripción Ideación De La Clase.

```cpp
class OthelloGame {
public:
    OthelloGame();
    ~OthelloGame();
    void play();

private:
    char **board;
    char currentPlayer;

    void initializeBoard();
    void displayBoard();
    bool isValidMove(int row, int col);
    void makeMove(int row, int col);
    bool isGameOver();
    void switchPlayer();
};
```

Ilustración 2. Primera Ideación

```cpp
class OthelloGame {
public:
    OthelloGame(); //Constructor de la clase, tablero NxN
    ~OthelloGame(); //Destrcutor que libera la memoria dinamica
    void play(); //Funcion que unifica todo e inicia el juego

private:
    char **board; //Tablero
    char currentPlayer; //Jugador del turno
    char oppositePlayer;
    bool hasValidMove;  //Varaibles auxiliares
    int consecutivePasses;

    void initializeBoard(); //Asignar las posiciones iniciales del juego al tablero
    void displayBoard(); //Mostrar el tablero
    bool isValidMove(int row, int col); //Validar el movimiento
    void makeMove(int row, int col); //Realizar el movimiento
    bool isGameOver(Data); //Funcion que comprueba si el juego termina o no
    void switchPlayer(); //Camabio de fichas
    int countTiles(char player); //Contador de fichas
    //bool isEnclosed(int row, int col, int rowDir, int colDir);
    string generateValidMoves(); //Guardar los movimientos validos
};
```

Ilustración 3. Ideación final

Clase Data

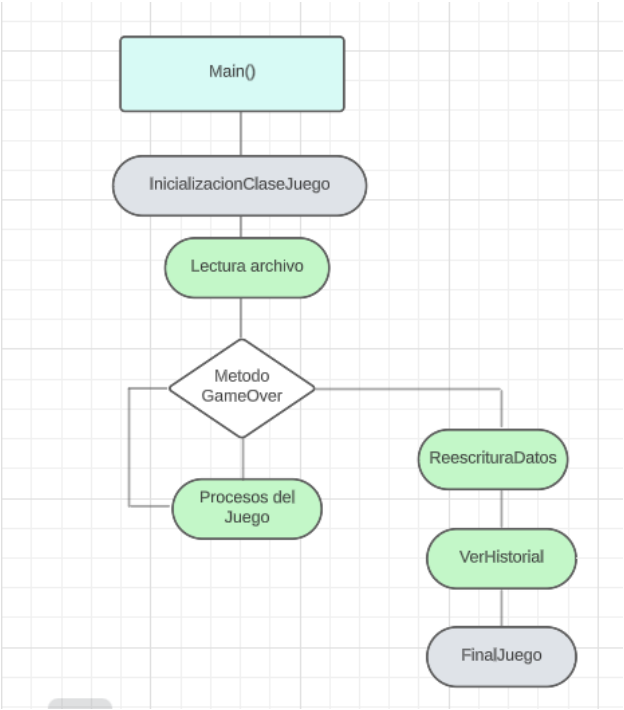
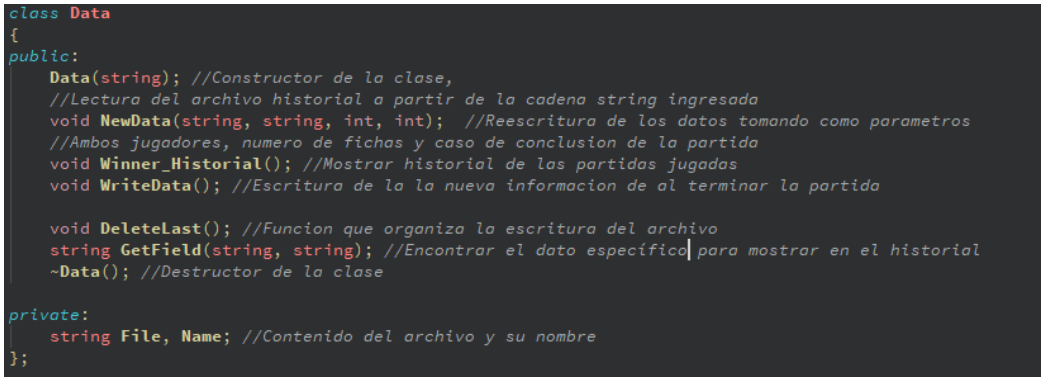


Ilustración 4. Diagrama De Flujo – Lógica de clase dentro de Juego

Descripción Ideación de la Clase.

```
class Data
{
public:
    Data(const char* fileName);  //Constructor de la clase, a partir del nombre del archivo tomamos su contenido
    void WriteData(); //Reescritura de los datos
    void Winner_Historial();
    ~Data();  //Eliminar contenido
private:
    const char* File; //Contenido del archivo
};
```

Ilustración 5. Primera Ideacion



```
class Data
{
public:
    Data(string); //Constructor de la clase,
    //Lectura del archivo historial a partir de la cadena string ingresada
    void NewData(string, string, int, int);  //Reescritura de los datos tomando como parametros
    //Ambos jugadores, numero de fichas y caso de conclusion de la partida
    void Winner_Historial(); //Mostrar historial de las partidas jugadas
    void WriteData(); //Escritura de la la nueva informacion de al terminar la partida

    void DeleteLast(); //Funcion que organiza la escritura del archivo
    string GetField(string, string); //Encontrar el dato específico para mostrar en el historial
    ~Data(); //Destructor de la clase

private:
    string File, Name; //Contenido del archivo y su nombre
};
```

Ilustración 6. Ideacion Final

Algoritmos Implementados

https://github.com/JDiego11/Parcial-2/tree/0c5e4539bca003527656cbf4613325405bd07209/Desarrollo%20final/Juego/Parcial2

Experiencia de aprendizaje

1. Optamos por utilizar datos de tipo string en lugar de datos char para el manejo de la clase Data, y esta elección se basa en la notable facilidad que ofrecen los strings. Al utilizar datos de tipo string podemos tener una mayor versatilidad para el almacenamiento de información y su facilidad de manipulación debido a la simplificación de operaciones de lectura, escritura y procesamientos.
2. Enfrentamos una dificultad al intentar mostrar el tablero de manera amigable en la consola. Esta problemática se hacía evidente a medida que el tamaño del

tablero, representado por N, crecía. Nos encontramos con dos posibles abordar este problema.

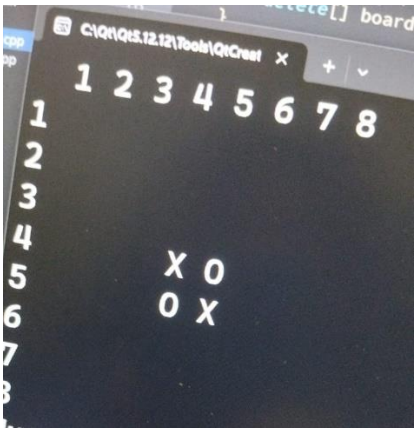
3. 

Ilustración7. Opcion 1 de Visualizacion

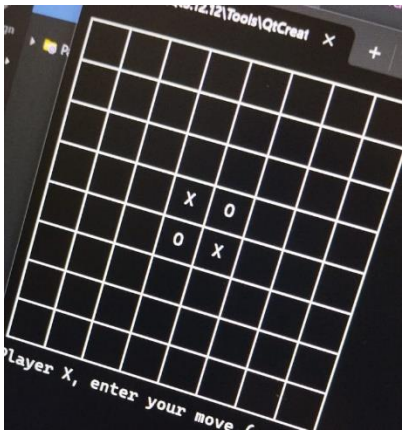


Ilustración 8. Opcion 2 Visualizacion

La opción número 1, aunque nos permite mostrar directamente el número de columnas y filas del tablero, presentaba inconvenientes a partir de cierto tamaño N. La presentación visual de la cuadrícula se volvía menos amigable y organizada, lo que podía dificultar la experiencia de juego.

La opción 2, por otro lado, nos brindaba la posibilidad de mostrar el tablero mediante una cuadrícula sin numeración. Aunque esta alternativa eliminaba la información numérica, permitía mantener una presentación más ordenada y fácil de entender, incluso a medida que el tamaño N aumentaba.

Finalmente, hemos optado por la opción 2, ya que consideramos que ofrece una solución más adecuada para evitar desorganización y mantener una interfaz intuitiva durante el juego, a pesar de la falta de numeración en el tablero. Esto garantiza una experiencia de juego más satisfactoria para los usuarios.