Informe Laboratorio 3
Paradigma orientado a objetos

Nombre: Ian Leonardo Rioseco Castro
Rut: 21.249.591-3
Profesor: Edmundo Leiva

Contenido

Introducción

En este informe, desarrollaremos como se construyó el juego conecta 4 utilizando el lenguaje de programación java, aplicando el paradigma orientado a objetos (POO). Conecta 4 es un juego para dos jugadores que se juega en un tablero vertical de 6 filas y 7 columnas [anexo 1]. Cada jugador tiene un color asociado, “rojo” o “amarillo”, y cada jugador tiene 21 fichas para toda la partida. Los jugadores alternan turnos para dejar caer sus fichas en las columnas del tablero, buscando formar una línea de 4 fichas consecutivas del mismo color en sentido vertical, horizontal o diagonal, Si el tablero se llena o ninguno logra formar una línea de cuatro, el juego termina en empate.

El informe esta estructurado por los siguientes capítulos.

- Introducción: Presenta el contexto y objetivo del proyecto.
- Análisis del problema: Desglosa el problema a resolver, sus características y limitaciones.
- Diseño de la solución: Evalúa el cumplimiento de los requisitos funcionales y las pruebas realizadas.
- Conclusión: Discute los beneficio y limitaciones del enfoque lógico aplicado al desarrollo del juego.

Descripción del problema

El problema principal es desarrollar una versión digital del juego conecta 4 implementada en prolog, que permita simular todas las acciones posibles del juego físico: colocar fichas, verificar victorias o empates y gestionar los turnos de los jugadores.

Conecta 4 es un juego competitivo para dos jugadores que se desarrolla sobre un tablero de 6 filas por 7 columnas. Los jugadores alternan turnos insertando fichas de su color en una de todas las columnas disponibles, Estas fichas “caen” hasta la posición mas baja vacía de la columna seleccionada. El objetivo es ser el primero en formar una línea de 4 fichas consecutivas de su color en cualquier dirección, ya sea vertical, horizontal o diagonal [anexo 2].

Características del problema:

- Dimensiones del problema: El tablero tiene una estructura fija de 6 filas por 7 columnas, con un máximo de 42 fichas jugables.
- Reglas básicas: Los jugadores se alternan los turnos insertando una ficha, el juego termina cuando se cumple con alguna de estas condiciones:
    - Un jugador logra formar una línea en cualquier dirección con 4 fichas consecutivas.
    - El tablero se llena y no existe ganador, por lo tanto, termina en empate.

- Limitaciones físicas del problema: una columna no puede aceptar mas fichas si todas sus filas están llenas.

Limitaciones del problema:

- El tablero esta limitado a un área de juego especifica, lo que permite una cantidad de movimientos predeterminada.
- Cantidad de fichas limitada.
- Cumplir con alguna de las condiciones de victoria para que haya ganador.

Descripción del paradigma

El paradigma orientado a objetos (POO) se basa en una estructura compuesta por clases y objetos, los cuales representan entidades del mundo real. Este enfoque organiza el programa en torno a conceptos como atributos (propiedades de los objetos) y métodos (acciones y comportamientos asociados). Su principio fundamental es la encapsulación, que asegura que los datos de cada objeto sean manipulados exclusivamente a través de sus métodos, promoviendo módulos y reutilización del código.

Un elemento clave en el paradigma orientado a objetos es el uso de herencia y polimorfismo, que permiten reutilizar y extender funcionalidades de manera eficiente. Este paradigma se adapta perfectamente a la implementación del juego conecta4, ya que facilita la modelación de entidades como jugadores, Fichas, Tablero y el juego en general, mediante el uso de clases que interactúan entre si a través de métodos bien definidos.

Análisis del problema

El problema por resolver en este proyecto es la implementación digital del juego Conecta 4 haciendo uso del paradigma orientado a objetos en el lenguaje de programación Java. Para resolver esto, será necesario diseñar e implementar clases que representen las entidades principales del juego, definir sus atributos y métodos, y establecer las interacciones entre los diferentes componentes. Esto nos ayudará a simular cada una de las operaciones que se pueden realizar en el Conecta 4 físico y a manejar las diferentes situaciones que pueden surgir durante el juego.

La creación de clases bien definidas será clave para mantener un orden y claridad en el proyecto. Cada clase representará una parte del estado del juego, como los jugadores, el tablero, las fichas y el historial de jugadas. Los métodos de estas clases permitirán encapsular y gestionar el comportamiento de cada entidad, facilitando la interacción y la resolución de situaciones propias del juego.

Cada método tendrá un propósito claro:

- métodos constructores: Inicializaran las entidades como jugadores, tablero, fichas y el juego en general.
- métodos selectores: Consultarán el estado del juego, y permitirán realizar operaciones como insertar fichas o verificar condiciones de victoria o empate.
- métodos modificadores: Permitirán actualizar el estado del juego, como cambio de turno, registrar movimientos y disminuir piezas restantes.

Clases Principales:

- Player: Clase que representa el perfil de un jugador, incluye atributos como ID del jugador, nombre, color de la ficha, victorias, derrotas, empates y piezas restantes.
- Piece: Clase que representa una ficha, incluye el color de la ficha como atributo principal.
- Board: Clase que representa el tablero del juego, que consta de 6 filas y 7 columnas. Cada posición dentro de la matriz es una celda del tablero.
- Game: Clase que representa la creación de una partida. Incluye atributos como los jugadores, tablero, turno actual y un historial de movimientos.

Diseño de la solución

Para abordar el desarrollo del juego Conecta 4, se proponen cuatro Clases principales que organizan y gestionan los elementos del juego, estas clases implementan su respectiva interfaz, además se implementó un menú interactivo para interactuar por terminal.

Interfaces:

- PlayerInterface: contiene todos los métodos necesarios para implementar un jugador, como consultar y actualizar estadísticas o fichas restantes, y consultar datos del jugador.
- PieceInterface: contiene los métodos para representar una ficha.
- BoardInterface: contiene los métodos para realizar una jugada y gestionar los estados del tablero y verificar las condiciones de victoria.
- GameInterface: contiene los métodos para crear un juego y obtener todos los estados de este, como manejar turnos, consultar el estado del juego y realizar jugadas.

Clases:

- Player: implementa la interfaz PlayerInterface. Representa el perfil del jugador. Almacena información como ID, nombre del jugador, color de la ficha, victorias, derrotas, empates y fichas restantes. Proporciona relaciones para consultar y actualizar estadísticas y fichas restantes, incluye un método constructor para actualizarlas.
- Piece: Implementa la interfaz PieceInterface. define la ficha de cada jugador mediante el color, permitiendo una clara diferenciación entre las fichas de ambos jugadores.
- Board: Representa el tablero como una “matriz” de guiones, organizando las filas y columnas necesarias. Incluyendo los métodos para verificar el estado del tablero, gestionar jugadas, colocar fichas en la posición más baja de cada columna y evaluar condiciones de victoria. Estas verificaciones se realizan mediante reglas que identifican cuatro fichas consecutivas del mismo color en cualquier dirección.
- Game: Controla el estado general del juego. Almacena los perfiles de ambos jugadores, el tablero actual, el turno en curso y un historial de movimientos de la partida. Ofrece métodos para verificar empates, actualizar estadísticas, determinar el jugador actual y finalizar el juego. También es posible manejar turnos, validar jugadas y actualizar el tablero e historial.
- Menú: consiste en 5 opciones, las cuales permiten:
    1. crear un juego, solicita los datos de los dos jugadores y configura el tablero.
    2. visualizar el estado actual del tablero, muestra el tablero en su estado actual, solo si ya se ha creado un juego.
    3. realizar una jugada, permite al jugador realizar una jugada en una columna especifica.
    4. ver estadísticas del juego, muestra las estadísticas actuales del juego, incluye victorias, derrotas y empates de los dos jugadores.

5. salir del juego, Finaliza la ejecución.

Consideraciones de implementación

El juego fue implementado en java utilizando las siguientes herramientas:

- Entorno de desarrollo: intellij IDEA 2024.1
- JDK: versión 21.

Cada clase e interfaz tiene su propio archivo dentro, para tener un mayor orden, y obtener un enfoque modular donde cada clase encapsula una parte del comportamiento del juego, las conexiones entre clases se realizan mediante los métodos.

Instrucciones de uso

Configuración inicial:

- Asegurarse de tener instalado java JDK 21.
- Importar el proyecto en un IDE como intellij o Eclipse.

Ejecución:

- Compila y ejecuta el Main.java.
- mainClass conecta4.Menu.
- Interactúa con el menú para crear un juego, realizar jugadas y consultar estadísticas. [Anexo 3]

Resultados obtenidos y Autoevaluación

En el proyecto se alcanzaron todos los requisitos funcionales pedidos 18/18, ninguno de los requerimientos funcionales pedidos no se alcanzó, de los 18 requisitos alcanzados todos tienen un 100% de alcance [anexo 4] debido a las pruebas realizadas ya que el 100% veces entregaban resultados correctos y que concordaban con las pruebas realizadas, lo que se utilizó para las pruebas fue el ejemplo de menú y flujo de ejecución entregado en el documento de instrucciones del laboratorio N3. La creación de las relaciones para verificar victorias resultó ser particularmente desafiante debido a la variedad de condiciones que debían considerarse, lo cual no afectó su fiabilidad en todos los casos posibles. Para abordar este desafío, se decidió encapsular la lógica de verificación de victorias en una clase específica, la cual contiene métodos para evaluar las diferentes condiciones de victoria. Esta estructura orientada a objetos permite una mejor organización del código y facilita la modificación de las condiciones en el futuro.

Además, se crearon 2 diagramas UML, uno antes de realizar el código [ANEXO 5] de cada clase y uno después de realizar el código [Anexo 6] de cada clase y menú, esto para observar como cambian estos diagramas. Pudiendo observar que el primero presenta un diseño inicial del sistema mientras que el segundo diagrama un refinamiento del diseño final, las principales diferencias incluyen la creación de las interfaces dentro del diseño final, lo que mejora la abstracción y facilita la reutilización del Código. Además, se agrega una nueva clase menú en el segundo diagrama para gestionar las interacciones del usuario. El diseño final también detalla parámetros en métodos como playpiece y realizarmovimiento, especificando el uso de columnas y colores. Finalmente, el atributó flag se incluye en Game el cual sirve para verificar si el juego está terminado o no.

Conclusión

El desarrollo del juego Conecta 4 utilizando el paradigma orientado a objetos (POO) en Java ha permitido abordar de manera estructurada y eficiente la implementación de un juego clásico como el conecta4 en un entorno digital. La elección del POO ha facilitado la creación de clases que representan las entidades del juego, como jugadores, piezas, tablero y la lógica de la partida. Este enfoque ha promovido el modularidad y la reutilización del código, permitiendo que cada componente del sistema tenga su propia responsabilidad y comportamiento definido.

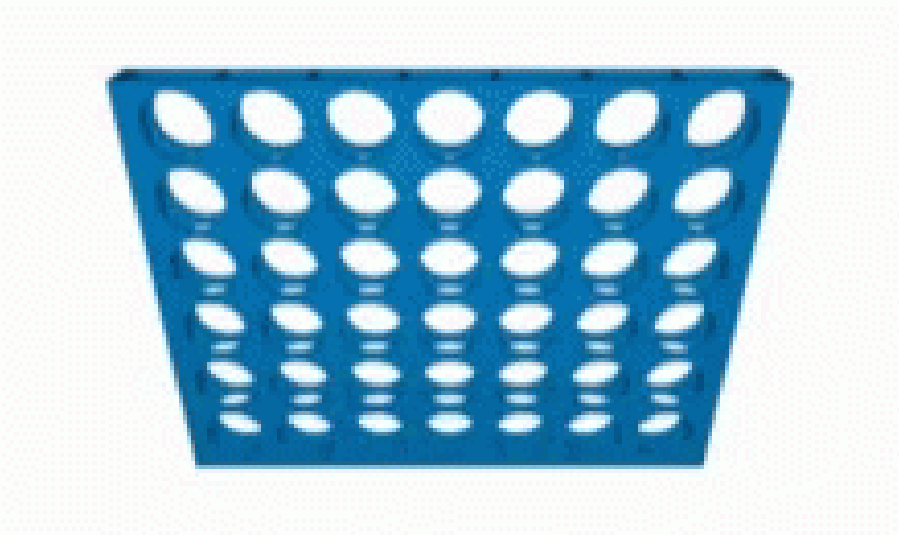
A lo largo del proyecto, se logró cumplir con todos los requisitos funcionales establecidos, alcanzando un 100% en las pruebas de funcionalidad. Esto no solo valida la solidez de la implementación, sino que también destaca la eficacia de la planificación y diseño inicial, que incluyó diagramas UML que guiaron el desarrollo. Las dificultades encontradas, especialmente en la verificación de las condiciones de victoria, fueron superadas mediante la encapsulación de esta lógica en una clase específica, lo que mejoró la organización y la mantenibilidad del código.

El diseño final, que incluye interfaces para cada entidad principal y un menú interactivo para la interacción del usuario, representa una mejora significativa respecto al diseño inicial. Esta evolución no solo mejoró la abstracción del sistema, sino que también facilitó la ampliación y modificación de las características del juego en el futuro.

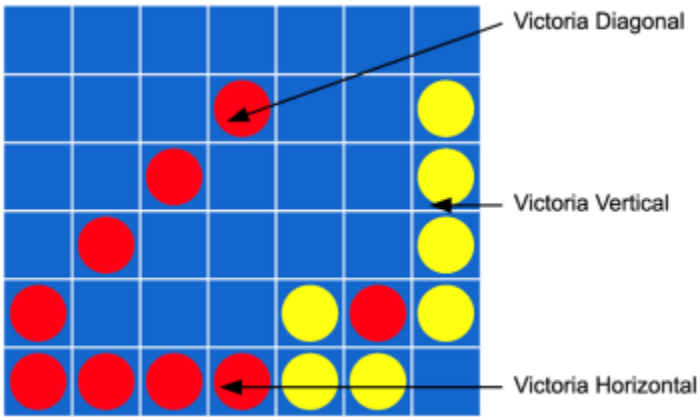
En conclusión, este proyecto no solo ha sido un ejercicio técnico, sino también una oportunidad para profundizar en los principios del POO y su aplicación práctica en un contexto de desarrollo de software. La experiencia adquirida en el proceso de diseño, implementación y prueba de este juego digital de Conecta 4 ha fortalecido la comprensión de los conceptos fundamentales de la programación orientada a objetos, así como la importancia de una planificación adecuada y un diseño claro para el éxito de cualquier proyecto de desarrollo de software.

Anexos

1) Anexo 1



2) Anexo 2



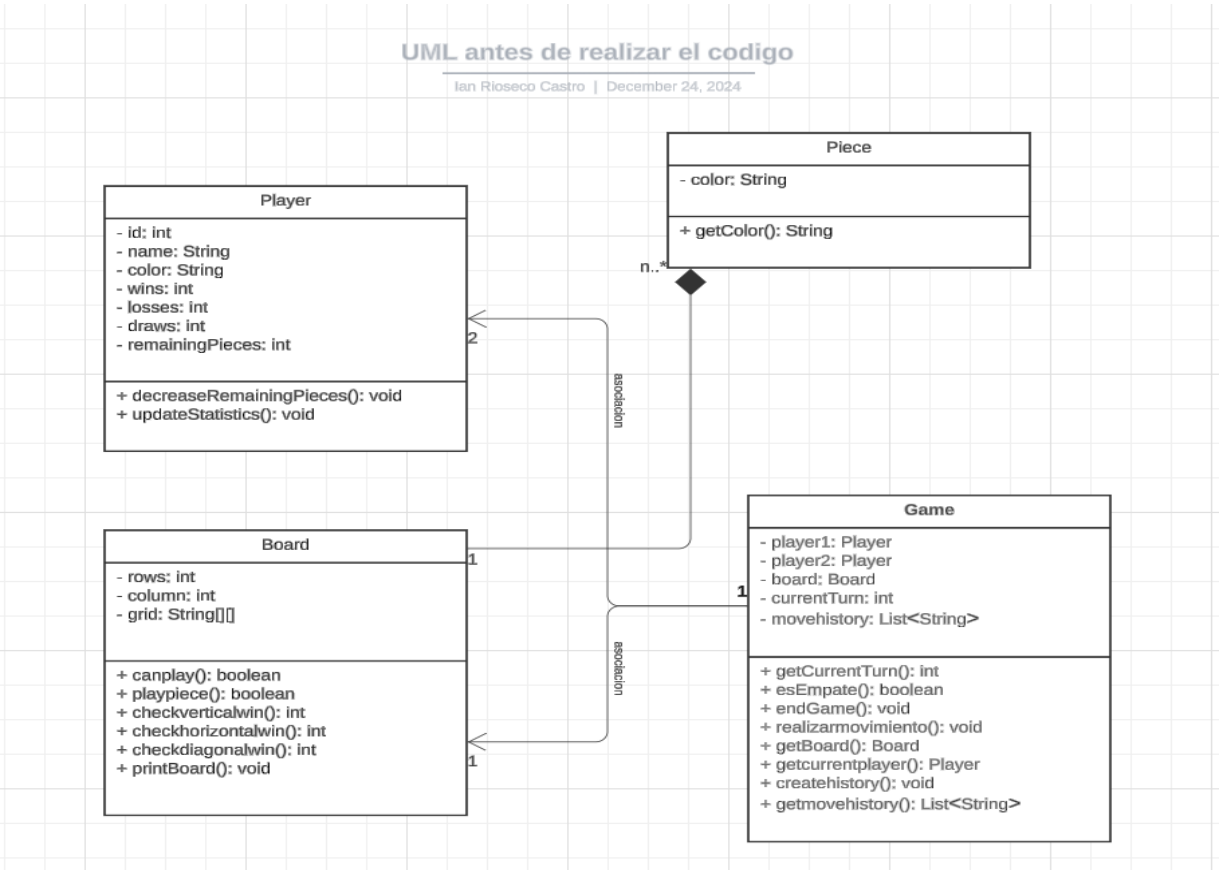
3) Anexo 3

```
### Conecta4 - Menú Principal ###
1. Crear nuevo juego
2. Visualizar estado actual
3. Realizar jugada
4. Ver estadísticas generales
5. Salir del juego
```

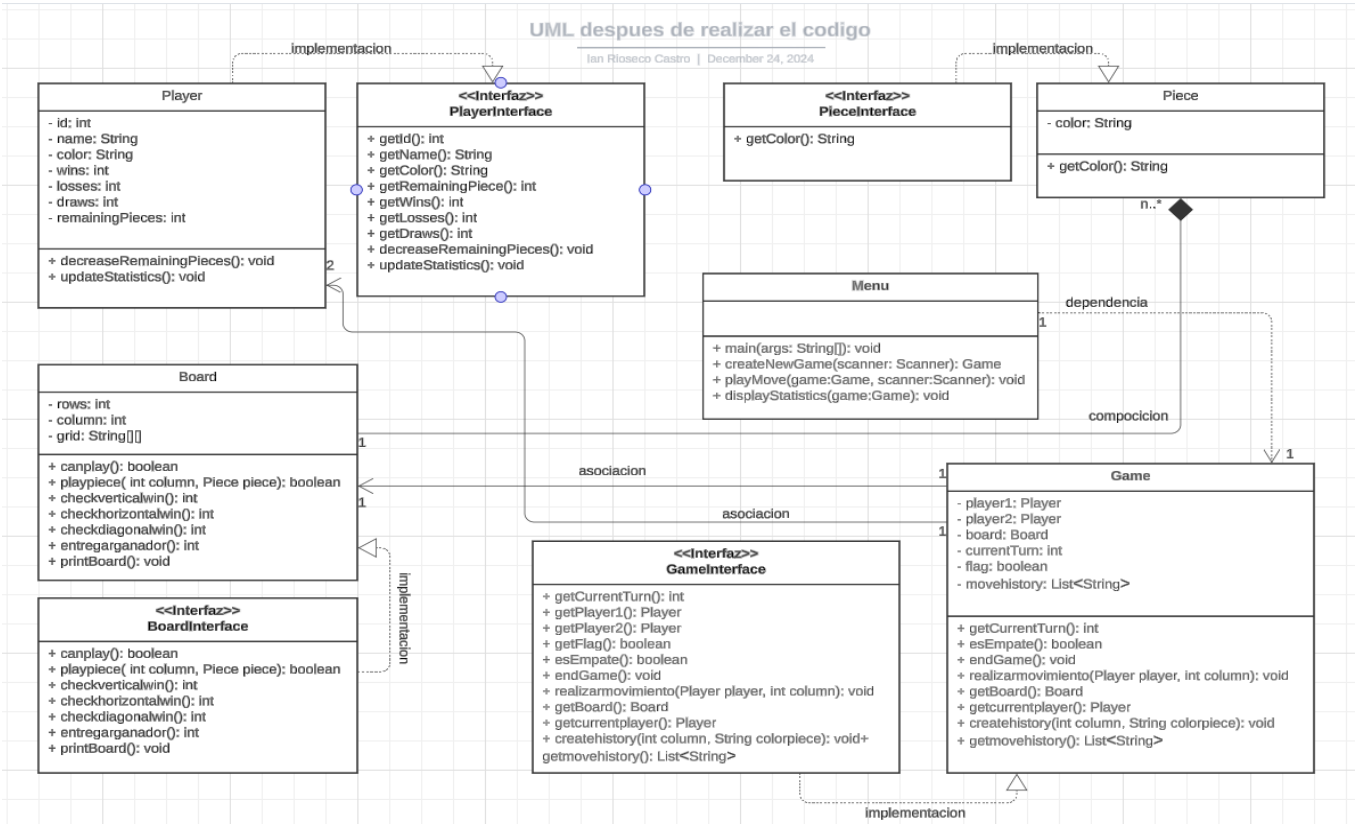
4) Anexo 4

```
Escala:
0: No realizado.
0.25: Implementación con problemas mayores (funciona 25% de las veces o no funciona)
0.5: Implementación con funcionamiento irregular (funciona 50% de las veces)
0.75: Implementación con problemas menores (funciona 75% de las veces)
1: Implementación completa sin problemas (funciona 100% de las veces)

TDA / Función / Puntaje
TDA Player-constructor / player / 1
TDA Player-selector / getId / 1
TDA Player-selector / getName / 1
TDA Player-selector / getColor / 1
TDA Player-selector / getWins / 1
TDA Player-selector / getLosses / 1
TDA Player-selector / getDraws / 1
TDA Player-selector / getRemainingPieces / 1
TDA Player-otros / updateStatistics / 1
TDA Player-otros / decreaseRemainingPieces / 1

TDA Piece-constructor / piece / 1
TDA Piece-selector / getColor / 1

TDA Board-constructor / board / 1
TDA Board-otros / canplay / 1
TDA Board-modificador / playPiece / 1
TDA Board-otros / checkverticalwin / 1
TDA Board-otros / checkhorizontalwin / 1
TDA Board-otros / checkdiagonalwin / 1
TDA Board-otros / entregarganador / 1

TDA Game-constructor / game / 1
TDA Game-otros / createHistory / 1
TDA Game-otros / esEmpate / 1
TDA Game-selector / getCurrentTurn / 1
TDA Game-selector / getCurrentPlayer / 1
TDA Game-selector / getPlayer1 / 1
TDA Game-selector / getPlayer2 / 1
TDA Game-selector / game-get-board / 1
TDA Game-selector / getMoveHistory / 1
TDA Game-modificador / endGame / 1
TDA Game-modificador / realizarMovimiento / 1
```

5) Anexo 5



6) Anexo 6


UML despues de realizar el codigo
Ian Rioseco Castro | December 24, 2024
implementacion
implementacion
Player
- id: int
- name: String
- color: String
- wins: int
- losses: int
- draws: int
- remainingPieces: int
+ decreaseRemainingPieces(): void
+ updateStatistics(): void
<<Interfaz>>
PlayerInterface
+ getId(): int
+ getName(): String
+ getColor(): String
+ getRemainingPiece(): int
+ getWins(): int
+ getLosses(): int
+ getDraws(): int
+ decreaseRemainingPieces(): void
+ updateStatistics(): void
<<Interfaz>>
PieceInterface
+ getColor(): String
Piece
- color: String
+ getColor(): String
n..*
2
Menu
+ main(args: String[]): void
+ createNewGame(scanner: Scanner): Game
+ playMove(game:Game, scanner:Scanner): void
+ displayStatistics(game:Game): void
dependencia
1
Board
- rows: int
- column: int
- grid: String[][]
+ canplay(): boolean
+ playpiece(int column, Piece piece): boolean
+ checkverticalwin(): int
+ checkhorizontalwin(): int
+ checkdiagonalwin(): int
+ entregarganador(): int
+ printBoard(): void
compocicion
1
asociacion
1
1
asociacion
1
Game
- player1: Player
- player2: Player
- board: Board
- currentTurn: int
- flag: boolean
- movehistory: List<String>
+ getCurrentTurn(): int
+ esEmpate(): boolean
+ endGame(): void
+ realizarmovimiento(Player player, int column): void
+ getBoard(): Board
+ getcurrentplayer(): Player
+ createhistory(int column, String colorpiece): void
+ getmovehistory(): List<String>
<<Interfaz>>
BoardInterface
+ canplay(): boolean
+ playpiece(int column, Piece piece): boolean
+ checkverticalwin(): int
+ checkhorizontalwin(): int
+ checkdiagonalwin(): int
+ entregarganador(): int
+ printBoard(): void
implementacion
<<Interfaz>>
GameInterface
+ getCurrentTurn(): int
+ getPlayer1(): Player
+ getPlayer2(): Player
+ getFlag(): boolean
+ esEmpate(): boolean
+ endGame(): void
+ realizarmovimiento(Player player, int column): void
+ getBoard(): Board
+ getcurrentplayer(): Player
+ createhistory(int column, String colorpiece): void+
getmovehistory(): List<String>
implementacion

Bibliografía

SWI-Prolog. (s/f). Swi-prolog.org. Recuperado el 27 de mayo de 2024, de https://www.swi-prolog.org/

SWISH -- SWI-Prolog for SHaring. (s/f). Swi-prolog.org. Recuperado el 27 de mayo de 2024, de https://swish.swi-prolog.org/

(S/f). Edu.ar. Recuperado el 27 de mayo de 2024, de https://users.exa.unicen.edu.ar/catedras/prog_exp/apuntes/clase3.pdf