

UNIVERSIDAD DE SANTIAGO DE CHILE
FACULTAD DE INGENIERÍA
Departamento de Ingeniería Informática



CONECTA 4

PROYECTO DE LABORATORIO 3 – PARADIGMA ORIENTADO A OBJETOS

Paradigmas de Programación

Matías Fernando Ramírez Escobar 21.484.373-0

matias.ramirez.e@usach.cl

Sección: 13204-0 A-1

Fecha de entrega: 26/12/24

Profesor: Edmundo Leiva

**Carrera: Ingeniería de ejecución
en computación e informática**

Tabla de contenidos

Introducción.....	2
El problema	3
Scheme y su paradigma	3
Análisis del problema.....	4
Diseño de la solución.....	5
Aspectos de implementación	6
Instrucciones de uso.....	6
Resultados y autoevaluación	7
Conclusiones del trabajo.....	7
Referencias	8
Anexos.....	9

Introducción

En el campo de la programación, los paradigmas desempeñan un papel crucial al proporcionar enfoques diversos para resolver problemas de manera eficiente. Este informe aborda la implementación del juego Conecta 4 utilizando el lenguaje de programación Java y el paradigma orientado a objetos.

Desde esta perspectiva, se pone de relieve cómo el empleo de conceptos clave como clases, herencia, encapsulamiento y polimorfismo contribuye al desarrollo de un sistema estructurado, mantenible y escalable. La elección de Java, un lenguaje ampliamente adoptado en la industria asegura una base sólida y un enfoque práctico para abordar este desafío. En un mundo donde la vida digital se encuentra

cada vez más presente, la compañía DINFF, presentará de manera digital uno de sus juegos más populares, para lograr abarcar mayor cantidad de público.

El juego Conecta 4 tiene como objetivo principal promover en los niños el desarrollo de habilidades cognitivas, sensoriales, sociales y emocionales. Además, fomenta el razonamiento lógico, la imaginación y la creatividad, convirtiéndose en una herramienta didáctica que combina entretenimiento con aprendizaje.

El problema

El problema central de este proyecto es lograr desarrollar una versión interactiva y funcional del juego Conecta 4 utilizando Java. Este desarrollo abarca el diseño e implementación de un tablero que represente fielmente el juego, la gestión de las interacciones entre los jugadores y la validación de las reglas para garantizar una experiencia fluida y consistente.

Aprovechando las características del paradigma orientado a objetos que ofrece Java, el diseño se centra en la creación de clases específicas que modelen los componentes clave del juego: el tablero, los jugadores y las fichas. El juego Conecta 4 es únicamente para dos jugadores, en donde se utilizan de 4 a 21 fichas por jugador, es un juego de turnos el cual termina una vez que se conectan 4 fichas consecutivas de manera vertical, horizontal o diagonal. Este proyecto enfatiza la importancia de lograr un código modular y reutilizable, asegurando así una solución escalable, mantenible y adaptable a futuras mejoras.

JAVA y su paradigma

Java es un lenguaje de programación de propósito general, conocido por su portabilidad, seguridad y amplia gama de aplicaciones. Uno de sus pilares fundamentales es la programación orientada a objetos (POO), un paradigma que nos permite modelar el mundo real en términos de objetos que interactúan entre sí.

La POO se basa en el concepto de objetos. Un objeto es una entidad que encapsula tanto datos (atributos) como comportamientos (métodos). Por ejemplo, un coche es un objeto con atributos como color, marca y modelo, y métodos como acelerar, frenar y girar.

Ventajas de la POO:

- **Reutilización de código:** Las clases pueden ser reutilizadas en diferentes partes de un programa, lo que ahorra tiempo y reduce errores.
- **Mantenibilidad:** El código organizado en clases es más fácil de entender y modificar.
- **Escalabilidad:** Los sistemas diseñados con POO pueden crecer y adaptarse a nuevas necesidades de manera más sencilla.

- **Abstracción:** Permite modelar sistemas complejos de forma más sencilla, ocultando los detalles de implementación.

Conceptos Clave de la POO en Java

- **Clases y objetos:**
 - Una clase es como un plano o plantilla que define los atributos y métodos de un objeto.
 - Un objeto es una instancia de una clase, es decir, una copia concreta de esa clase con sus propios valores para los atributos.
- **Encapsulación:**
 - Consiste en ocultar los detalles internos de un objeto y proporcionar una interfaz bien definida para interactuar con él. Esto protege los datos internos de modificaciones accidentales y facilita el mantenimiento del código.
- **Herencia:**
 - Permite crear nuevas clases (subclases) a partir de clases existentes (superclases), heredando sus atributos y métodos. Esto fomenta la reutilización de código y la creación de jerarquías de clases.
- **Polimorfismo:**
 - Permite que objetos de diferentes clases puedan ser tratados como si fueran de la misma clase. Esto se logra a través de la sobrecarga y la sobrescritura de métodos.

Aplicación de la POO en Conecta 4:

- En Conecta 4, se utiliza la POO para modelar los diferentes elementos del juego, por ejemplo:
 - Clase Tablero: Representa el tablero de juego, con atributos como tamaño y métodos para comprobar si hay cuatro fichas en línea o si una columna está llena.
 - Clase Jugador: Representa a un jugador, con atributos como nombre y métodos para realizar movimientos.

Análisis del problema

El objetivo de este proyecto es desarrollar el juego Conecta 4 en Java utilizando el paradigma de programación orientada a objetos (POO). El juego está compuesto por tres elementos clave: jugadores, tablero y piezas. El sistema debe permitir crear una nueva partida, gestionar a los jugadores, y controlar el desarrollo del juego.

El programa debe permitir a los jugadores realizar movimientos alternados, validarlos y verificar si alguno cumple las condiciones para ganar, ya sea por una línea vertical, horizontal o diagonal. Además, debe gestionar las estadísticas de los jugadores, permitiendo registrar nuevos jugadores, modificar sus datos y consultar sus victorias.

El sistema también debe ser capaz de detectar si se ha producido una victoria, un empate o si el juego continúa. Para ello, el tablero debe actualizarse y mostrar su estado en cada turno, permitiendo a los jugadores ver las fichas en las posiciones correspondientes.

Se crea un diagrama UML (ver anexo 1) a partir de lo planteado con el propósito de modelar el problema en base a los requerimientos funcionales que debe tener este. En el diagrama se observan los atributos y métodos de cada clase y como estas se relacionan entre sí.

El uso de POO facilita la modularidad del código, permitiendo gestionar cada componente del juego de manera independiente, lo que simplifica su implementación y mantenimiento.

A continuación, se detallan los aspectos clave para simular una partida de Conecta 4 en Java:

1. Creación y gestión de partidas:
 - Iniciar partida: Definir los jugadores y establecer el estado inicial del tablero.
2. Gestión de Jugadores:
 - Registro de jugadores: Asignar nombres y colores de fichas a los participantes.
 - Actualización de información: Modificar datos de los jugadores mientras se desarrolla la partida.
3. Desarrollo del Juego:
 - Realización de movimientos: Implementar la colocación de fichas en el tablero respetando las reglas del juego.
 - Validación de movimientos: Verificar que los movimientos sean legales, considerando las posiciones válidas.
 - Detección de resultados: Identificar condiciones de victoria, empate o continuidad del juego.
 - Visualización del tablero: Mostrar el estado actual del tablero antes de cada movimiento.
4. Análisis del Juego:
 - Historial de movimientos: Registrar todas las jugadas realizadas en la partida, permitiendo un análisis posterior.

Mientras que las tres primeras categorías son esenciales para la simulación básica del juego, la funcionalidad de un historial de movimientos enriquece la experiencia del usuario. Este predicado permite a los jugadores revisar estrategias, analizar jugadas previas y optimizar su desempeño en partidas futuras, superando las capacidades del juego físico, además, al ser implementado en JAVA esta misma lógica de resolución para esta simulación puede llevarse en práctica a otros juegos como un tres en raya o cambiando las lógicas de victoria a unas damas chinas.

Este enfoque integral garantiza una representación lógica y completa del juego, alineada con las capacidades de JAVA y diseñada para ofrecer una experiencia interactiva enriquecedora.

Diseño de la solución

Para representar el juego Conecta 4 de manera efectiva, es esencial identificar los principales Tipos de Datos Abstractos (TDA) que serán implementados mediante diversas clases. Con este enfoque el proyecto se divide en tres TDA principales (ver anexo 3): el TDA tablero, el TDA jugador y el TDA juego, Para unificar todos los TDA y lograr hacer un menú funcional y didáctico se utilizó un nuevo TDA llamado Main el cual se encarga de completar la simulación y donde se pueden hacer los llamados para crear las estructuras pertinentes como un nuevo juego y poder operar en este.

Un aspecto importante del diseño es que, al iniciar el programa se mostrará por terminal las opciones que poseen los jugadores, al crear un nuevo juego pueden comenzar a jugar. Durante cada turno, el jugador debe especificar la columna en la que desea colocar su ficha, la cual siempre caerá en la última casilla disponible, de arriba hacia abajo. El programa actualizará el estado del juego tras cada movimiento. Estas actualizaciones incluyen:

- Colocar la ficha en el tablero.
- Cambiar el turno.
- Disminuir el número de fichas restantes para el jugador que realizó el movimiento.
- Actualizar el historial de movimientos.
- Verificar si hay un ganador o si el juego ha terminado en empate.
- Si el juego ha concluido, actualizar las estadísticas de ambos jugadores según corresponda.

Para lograr una implementación clara y eficiente, se emplearon diversas herramientas propias de JAVA, como la creación de interfaces y el uso de herencia, cada clase usada utiliza la encapsulación con el objetivo de lograr un menor acoplamiento además de que se desea apuntar al reciclaje de código por medio del polimorfismo, logrando o apuntando a lograr una alta cohesión.

Aspectos de implementación

Para la implementación de este proyecto se utilizó el IDE IntelliJ IDEA versión 2024.1 con un SDK preciso concretamente la versión Project SDK temurin-11(Eclipse Temurin 11.0.25), a demás se implementa la herramienta Gradle como medio de compilación. El equipo usado para la elaboración de este proyecto fue un notebook hp con las siguientes características: Intel(R) Core(TM) i5-1035G1, 8gb de memoria RAM y grafica integrada.

Estructura y Encapsulamiento:

- La clase game es el núcleo del juego y manipula las clases internas.
- Las clases board (tablero) y player (jugador) tienen metodos específicos que permiten manipular sus datos.

El modularidad de este enfoque permite que se pueda modificar o extender el sistema sin afectar las funcionalidades de los demás componentes, facilitando la mantenibilidad y la expansibilidad del código.

La documentación oficial de JAVA fue la principal fuente de referencia para comprender las características del lenguaje y optimizar el uso de sus funcionalidades.

Uso de Herramientas de IA:

- Al inicio del desarrollo, se utilizaron herramientas de Inteligencia Artificial para interpretar los errores de sintaxis y lograr adaptar la mente a este nuevo paradigma, no así en la escritura y decisión lógica del proyecto todo es hecho a mano.

Este enfoque de implementación asegura una estructura sólida y eficiente que facilita la creación, el análisis y la extensión del juego Conecta 4 en JAVA, manteniendo siempre la claridad y la legibilidad del código.

Instrucciones de uso

Este proyecto ha sido diseñado pensando en una fácil ejecución, Una vez descargados los archivos deben de seguir los siguientes pasos:

1. Necesita instalar SDK OpenJDK versión 11 (link) para su sistema operativo.
2. Abrir la carpeta "**Proyecto_21484373_MatiasRamirezEscobar**" (**anexo 4**).
3. Abrir la terminal en la carpeta y ejecutar el comando **.\gradlew.bat build** y posteriormente el comando **.\gradlew.bat run**.

Se espera que el programa funcione correctamente en todos los Requisitos Funcionales (RF), siempre y cuando el usuario no introduzca datos inválidos. De lo contrario, podrían generarse errores. Algunos ejemplos de situaciones que pueden causar errores incluyen entradas no esperadas por ejemplo colocar un entero en vez de un string o biceversa.

Resultados y autoevaluación

Los resultados obtenidos respecto a los Requisitos Funcionales (RF) y Requisitos No Funcionales (RNF) se presentan detalladamente en los Anexos 6 en adelante. No obstante, de manera general, se puede afirmar que el programa funciona correctamente, se cambiaron los comportamientos de las clases del diagrama original como se puede ver en el anexo 2.

Bajo estas condiciones, el proyecto ha demostrado ser exitoso en la implementación del juego Conecta 4 utilizando el paradigma orientado a objetos. El código cumple con los objetivos planteados inicialmente, logrando una simulación completa del juego sin presentar errores. Esto confirma que las técnicas empleadas en el diseño y desarrollo del proyecto, como el manejo de Tipos Abstractos de Datos (TDA) y la recursión, fueron adecuadas para modelar de manera efectiva la lógica del juego en JAVA.

En cuanto a la autoevaluación, se considera que se ha logrado cumplir con todos los Requisitos Funcionales esenciales, garantizando un desarrollo estable y eficiente. Los elementos implementados cumplen con las expectativas iniciales del proyecto, ofreciendo una experiencia de juego fluida y sin errores, siempre y cuando los usuarios sigan correctamente las instrucciones de uso. La simulación del juego es precisa, y todas las interacciones entre los jugadores se manejan de manera coherente, respetando las reglas del juego.

Sin embargo, se identificaron algunos aspectos a mejorar. El manejo de errores en entradas incorrectas podría ser más robusto, ya que, aunque el juego funciona correctamente bajo condiciones ideales, se podría mejorar la retroalimentación en caso de fallos. Por último, el juego actual se basa completamente en un sistema de interacción por consola. Para futuras versiones, podría implementarse una interfaz gráfica GUI o un sistema de representación visual del tablero para mejorar la experiencia del usuario.

El proyecto ha sido un éxito en cuanto a su implementación técnica y funcionalidad. Las decisiones de diseño, especialmente el uso de JAVA y la estructuración por medio de clases, han permitido lograr una solución eficiente y efectiva para modelar el juego Conecta 4 dentro del POO.

Conclusiones del trabajo

El presente proyecto representó un reto técnico significativo al implementar el juego Conecta 4 en Java utilizando el paradigma de programación orientada a objetos (POO). La adopción de este enfoque

permitió una organización del código de manera modular y escalable, lo que facilitó la gestión de los diferentes componentes del juego. A través de la planificación anticipada y un diseño estructurado, se logró que el sistema fuera capaz de abordar las funcionalidades clave del juego, como la detección de victorias, la validación de movimientos y la gestión de turnos.

Uno de los puntos fuertes del diseño fue la encapsulación, que permitió aislar los detalles de implementación interna de cada clase y exponer solo las interfaces necesarias para interactuar con otros componentes. La separación de responsabilidades también fue esencial para asegurar que cada clase tuviera una función clara y bien definida, evitando así la complejidad innecesaria y facilitando el mantenimiento y la extensión del código.

El paradigma de la programación orientada a objetos resultó ser altamente eficaz para gestionar la complejidad inherente a un proyecto de este tipo. Sin embargo, también se evidenció que un diseño cuidadoso y la prevención de acoplamientos excesivos entre clases son fundamentales para garantizar la claridad en las interacciones entre los componentes. A pesar de que se logró cumplir con los objetivos establecidos, se identificaron oportunidades para mejorar, como la posible integración de una interfaz gráfica de usuario (GUI) para mejorar la experiencia del jugador, así como la incorporación de pruebas automatizadas que robustecieron aún más la confiabilidad del sistema.

En resumen, el desarrollo del proyecto reafirmó la relevancia de aplicar principios sólidos de ingeniería de software, tales como modularidad, reutilización y diseño claro. Esto permitió obtener una solución funcional y extensible, que no solo cumple con los requisitos iniciales, sino que también sienta las bases para proyectos futuros de mayor envergadura y complejidad.

Referencias

Documento-laboratorio-3. (n.d.). *Documentación compartida: Paradigma orientado a objetos*. Retrieved November 28, 2024, from <https://docs.google.com/document/d/12PIbVd4w8kYq8z1xiSjfYIpDNEGDabclgtSbZQOCb8s/edit?tab=t.0>

GitHub. (2024). *Build and ship software on a single, collaborative platform*. GitHub. <https://github.com/>

Oracle. (n.d.). *Java documentation*. Oracle. <https://docs.oracle.com/en/java/>

Universidad de Santiago de Chile. (n.d.). *Plataforma Uvirtual: Curso 10036*. Universidad de Santiago de Chile. <https://uvirtual.usach.cl/moodle/course/view.php?id=10036§ion=20>

Anexos

1.-

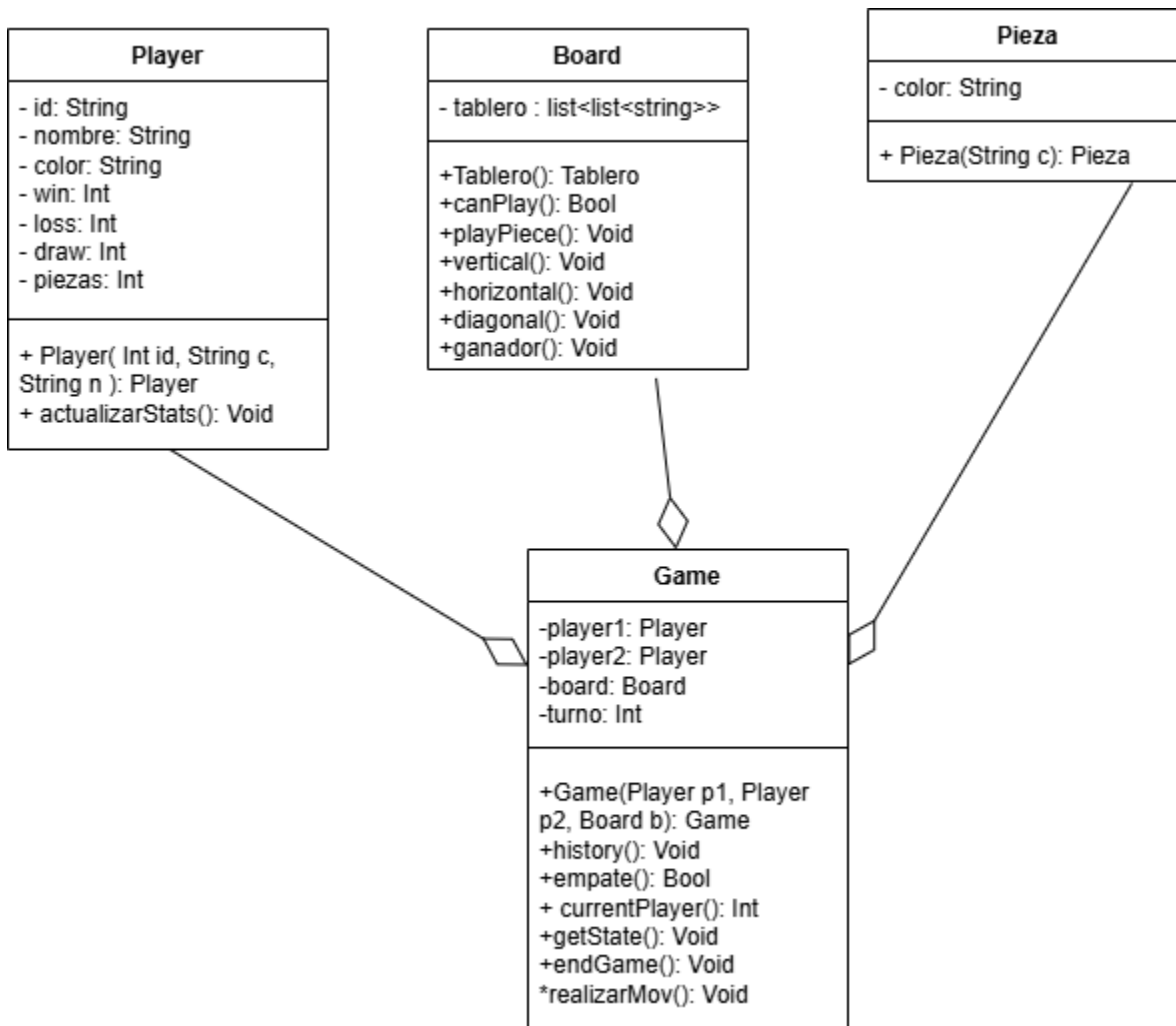


Diagrama UML antes de la implementación.

2.-

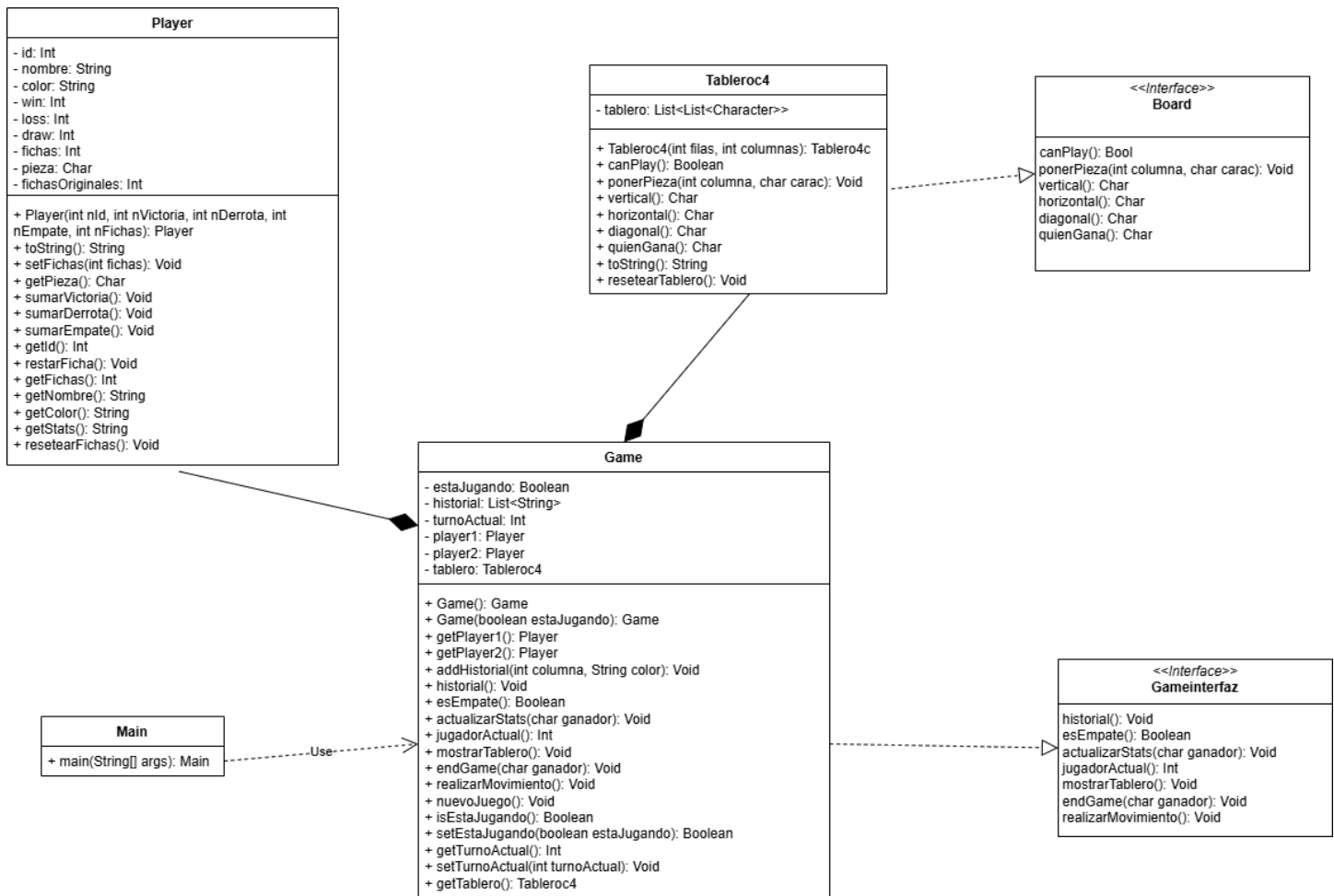













Diagrama UML luego de la Implementación

3.-

Nombre TDA	Dato Abstracto	Atributos	Métodos relevantes
Tablero	Representación de un tablero donde se colocarán las fichas y se usara para verificar ganadores u empate.	-tablero: List<List<Char>>	+Tableroc4(int filas, int columnas): Tablero4c +canPlay(): Boolean +ponerPieza(int columna, char carac): Void +vertical(): Char +horizontal(): Char +diagonal(): Char +quienGana(): Char +toString(): String +resetearTablero(): Void

Player	Representación de un jugador con sus propios atributos, además de poder actualizar estadísticas en caso de victoria derrota o empate.	-id: Int -nombre: String -color: String -win: Int -loss: Int -draw: Int -fichas: Int -pieza: Char -fichasOriginales: Int	+Player(int nId, int nVictoria, int nDerrota, int nEmpate, int nFichas): Player +toString(): String +setFichas(int fichas): Void +getPieza(): Char +sumarVictoria(): Void +sumarDerrota(): Void +sumarEmpate(): Void +getId(): Int +restarFicha(): Void +getFichas(): Int +getNombre(): String +getColor(): String +getStats(): String +resetearFichas(): Void
Game	Representación del juego conecta 4, la cual se forma a partir del conjunto de este TDA y los anteriores descritos, el cual puede colocar una ficha y hace las respectivas verificaciones necesarias para el correcto funcionamiento del programa.	-estaJugando: Boolean -historial: List<Str> -turnoActual: Int -player1: Player -player2: Player -tablero: Tableroc4	+Game(): Game +Game(boolean estaJugando): Game +getPlayer1(): Player +getPlayer2(): Player +addHistorial(int columna, String color): Void +historial(): Void +esEmpate(): Boolean +actualizarStats(char ganador): Void +jugadorActual(): Int +mostrarTablero(): Void +endGame(char ganador): Void +realizarMovimiento(): Void +nuevoJuego(): Void +isEstaJugando(): Boolean +setEstaJugando(boolean estaJugando): Boolean +getTurnoActual(): Int +setTurnoActual(int turnoActual): Void +getTablero(): Tableroc4

Tabla de TDAs implementados.

 .gradle	26-12-2024 22:38	Carpeta de archivos	
 .idea	26-12-2024 22:42	Carpeta de archivos	
 build	26-12-2024 22:39	Carpeta de archivos	
 gradle	26-12-2024 22:38	Carpeta de archivos	
 JAVADOC	26-12-2024 22:38	Carpeta de archivos	
 src	26-12-2024 22:38	Carpeta de archivos	
 .gitignore	15-12-2024 20:57	Documento de te...	1 KB
 build.gradle	15-12-2024 22:35	Archivo de origen ...	1 KB
 gradlew	15-12-2024 20:57	Archivo	8 KB
 gradlew.bat	15-12-2024 20:57	Archivo por lotes ...	3 KB
 settings.gradle	15-12-2024 20:57	Archivo de origen ...	1 KB

4.-

Carpeta que contiene el proyecto de java.

```

BUILD SUCCESSFUL in 2s
5 actionable tasks: 5 executed
PS C:\Users\matia\Desktop\Paradigmas\Lab3_214843730_MatiasRamirezEscobar\Proyecto_21484373_MatiasRamirezEscobar> .\gradlew.bat run

> Task :run
***** BIENVENIDO A CONECTA 4 *****

SELECCIONE UNA OPCIÓN:
1- Crear Un Nuevo Juego.
2- Salir Del Juego.

Ingrese su opción:
<=====--> 75% ***** CREAR NUEVO JUEGO *****

***** Creando Jugador 1 *****
Ingrese El Nombre Del Jugador 1 :
<<=====--> 75% EXECUTING [9sColor Asignado: Rojo
El Player 1 mati Fue Creado Exitosamente
***** Creando Jugador 2 *****
Ingrese El Nombre Del Jugador 2 :
<=<=====--> 75% EXECUTING Color Asignado: Amarillo
El Player 2 jose Fue Creado Exitosamente
Defina El Número De Fichas (4-21):
<=====--> 75% EXECUTING [16s] ***** TABLERO NUEVO *****
[ - | - | - | - | - | - | - ]
[ - | - | - | - | - | - | - ]
[ - | - | - | - | - | - | - ]
[ - | - | - | - | - | - | - ]
[ - | - | - | - | - | - | - ]

* JUEGO CREADO EXITOSAMENTE *
Presiona cualquier tecla para comenzar a jugar....
<=====--> 75% EXECUTING [19s]
> :run

```

5.-
Ejemplo de correcta ejecución del programa.

6.-

Requerimientos	Grado de implementación	Pruebas realizadas	Pruebas exitosas %	Pruebas fracasadas %	Anotación
TDA's	1	4 TDA's implementados	100	0	
TDA Player - constructor	1	20 creaciones de player	100	0	Se crearon los jugadores al ir probando el código y desde el

					inicio resultado fructífero
TDA Piece - constructor	1	38 creaciones de pieza	100	0	El constructor no es implementado como tal ya que no hay una clase llamada pieza sin embargo se implementa como un atributo de player
TDA Board - constructor	1	35 creaciones de tablero	100	0	El tablero se crea mediante iteraciones.
TDA Board - otros - can_play	1	8 consultas	100	0	
TDA Board - modificador - play_piece	0,75	35 fichas colocadas	100	0	Se uso la estrategia de iterar desde la última fila hasta arriba, no existieron intentos fallidos
TDA Board - otros - check_vertical_win	1	25 consultas	90	10	Se verifica que el método funcione correctamente jugando por consola y solo al inicio por errores en la iteración no funciono, pero se logra implementar
TDA Board - otros - check_horizontal_win	1	30 consultas	100	0	Al implementar el método para victoria horizontal también se prueba por consola, pero en esta ocasión no hay errores en la iteración nunca
TDA Board - otros - check_diagonal_win	1	35 consultas	100	0	Tampoco existen errores gracias a

					estar familiarizado con la iteración
TDA Board - otros - who_is_winner	1	30 consultas	100	0	No existe ninguna falla al implementar, solo se usan condicionales básicas if-else
TDA Game - constructor	1	20 games construidos	100	0	
TDA Game - otros - game_history	1	15 llamadas	100	0	Se itera sobre una lista, fácil implementación.
TDA Game - otros - is_draw	1	20 verificaciones	100	0	Simple, con iteración y verificación de if else
TDA Player - otros - update_stats	1	15 actualizaciones	100	0	Utilización de getters según corresponde
TDA Game - selector - get_current_player	1	12 llamados al getter	100	0	Se utiliza una operación matemática de paridad, siempre resultado fructífero
TDA Game - selector - game_get_board	1	13 llamadas al getter	100	0	
TDA Game - modificador - end_game	1	30 términos de un juego	100	0	
TDA Game - modificador - player_play	1	36 veces que se juega	100	0	

Autoevaluación de requerimientos funcionales

7.-

RNF	Implementacion	Anotaciones
Autoevaluación	1	Presente en la carpeta del proyecto.
Lenguaje	1	Lenguaje requerido JAVA.
Versión	1	JAVA 11, idle intellij 2024.1.
Standard	1	Librería estándar.

Documentación	1	Documentado en Java doc. Se documenta el código correctamente
Interacciones	1	Se ejecuta por consola
Uso del paradigma	1	Se usan las herramientas que ofrece el paradigma
Pre-requisitos	1	Se respetan los pre-requisitos
Diagramas	1	Se realizan los dos diagramas UML
Uso de git	1	Se respetan los 10 commit mínimos

Autoevaluación de los requerimientos no funcionales