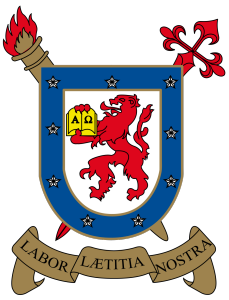
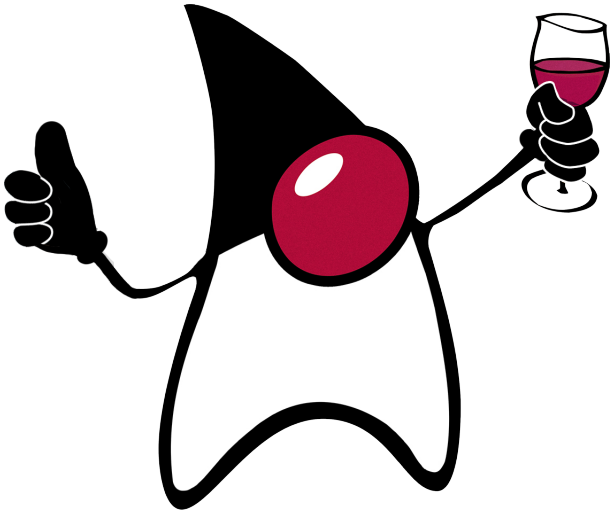


Laboratorio 3
Paradigmas de programación

Programación Orientada a Objetos - Java

Facultad de Ingeniería - Universidad de Santiago de Chile



Estudiante: Mateo Valle Lacourt
Profesor: Gonzalo Martinez Ramirez

Índice

1. Introducción

Conecta-4 es un famoso juego de mesa, en el cual dos jugadores alternadamente deberán colocar sus fichas en un tablero para conectar cuatro fichas consecutivas.

En esta actividad de laboratorio se planteó la creación de un programa que simule este juego. Dicho programa es hecho en el lenguaje de programación Java bajo el paradigma de programación orientado a objetos.

Primero se abordará el contexto del paradigma y el problema, luego se explicará el diseño de solución, también se darán instrucciones para poder jugarlo y se finalizará con una conclusión.

2. Descripción breve del problema

Conecta-4 es un juego de mesa de dos jugadores, los cuales deberán alinear cuatro fichas seguidas de forma vertical, horizontal o diagonal en un tablero posicionado de forma vertical de seis filas y siete columnas. El juego se basa en turnos, en los cuales un jugador coloca una ficha en cualquier columna del tablero y la ficha baja hasta lo mas abajo de este. Los turnos se van alternando hasta que se hayan conectado cuatro fichas seguidas, el cual será el ganador del juego o que no haya espacio en el tablero, lo cual cuenta como empate.

3. Descripción breve del paradigma

El paradigma orientado a objetos se basa en objetos, los cuales representan instancias del mundo real, tienen atributos(características físicas) y métodos(acciones que realizan), los objetos pueden interaccionar con otros objetos, creando cadenas de eventos, igual que en la vida real.

Para este laboratorio solo se verán ciertos conceptos del paradigma, que fueron los usados para la implementación del juego. Las clases son moldes para los objetos, de este mismo, se pueden crear objetos, con los atributos y métodos de la clase. La interface impone un orden en los métodos de las clases que la implementen, haciendo que varias clases tengan el mismo método, pero con implementaciones diferentes. Gracias a esto, se puede compartir un mismo fin para un método compartido por diferentes clases. En la interface solo se declaran los métodos, sin implementarlos, a estos métodos se les llama métodos abstractos. Las interacciones de objetos usados en este laboratorio son los siguientes:

Dependencia, cuando un objeto usa de forma temporal a otro.

Agregación, cuando un objeto A utiliza de forma indefinida a B, pero la vida de B no depende de A.

Composición, cuando un objeto A utiliza de forma indefinida a B, pero la vida de B depende estrictamente de A.

Implementación, cuando una clase hace uso de una interface.

También se utiliza el concepto de encapsulación, dicta que los atributos de los objetos sean privados, y que solo se puedan acceder a ellos mediante métodos selectores. Esto se usa para evitar un posible filtrado de datos innecesarios para los demás objetos.

4. Análisis del problema

La creación de un juego es complejo, por lo que se debe dividir en diferentes sub-problemas, que se complementen para dar una solución adecuada al problema planteado. En este proyecto, el juego se dividió en cuatro principales clases: Jugadores, Piezas, Tablero y Juego.

El jugador debe contener información acorde a este, como su identificador, un nombre, el color y cantidad de fichas y estadísticas de victorias, derrotas y empates. Su utilidad es la de brindar información al usuario y a los demás aspectos de implementción sobre los participantes del juego.

El tablero contiene las fichas de los jugadores, con estas se pueden realizar algunos de los diferentes tipos de verificaciones, tales como verificar victoria de forma vertical, horizontal y diagonal. Y también usar esas tres en combinación para entregar el posible ganador de la partida. También se debe implementar la función de colocar pieza y que quede en la posición mas baja posible, dejando solo espacios libres arriba de esta. Y por ultimo la verificación de tablero lleno, que dice si se puede jugar o no en el tablero.

El juego contiene todos los elementos antes mencionados, teniendo a los dos jugadores participantes, el tablero actualizado de la partida, el turno actual de jugador, un historial de movimientos realizados y un detector de estado de juego, el cual dice si el juego esta terminado o si aún se puede seguir jugando.

El conjunto de todos estos elementos conforman una solución al juego de Conecta-4, y cada uno de estos elementos se necesitan entre si para lograr implementar cada función del juego.

5. Diseño de solución

Antes del desarrollo de software, se hizo un diagrama de clases UML [1] pensando en una posible solución, se puede ver que es un diagrama bastante pobre en contenido, este se hizo en una etapa muy temprana del proyecto. Sin embargo, al finalizar el código, se realizo otro diagrama, esta vez con todas las caracteristicas funcionales del juego [2].

Se emplearon interfaces que representan un TDA para cada sub-problema presentado en el item 4 del informe. En dichas interfaces se declaran los métodos necesarios a realizar para las clases que los implementen, tales como los selectores, modificadores y otros métodos. no se pudo declarar los métodos constructores en la interface, ya que estas son especiales de las clases y son métodos instanciadores, osea que no pueden quedar declarados.

La clase Player implementa la interface Tda_Player, Player contiene su nombre, estadisticas, cantidad de piezas, un color y un id, que se obtiene a partir del atributo estático count_id, el cual es un entero que va sumando cada vez que se instancia un jugador, de esta forma, se le asigna un id a un jugador automáticamente al momento de instanciarlo. Player implementa los métodos de su TDA, tales como los selectores de sus atributos, modificadores como substraer una pieza, mostrar y actualizar sus estadísticas.

La clase Piece implementa la intertface Tda_Piece, Piece solo contiene un String que representa una ficha en el tablero e implementa solo al método para obtener dicho atributo.

La clase Board implementa la interface Tda_Board, Board contiene un tablero, representado como un arreglo de dos dimensiones, también realiza composición con Piece, ya que Board instancia a dicho objeto y se guarda como dos atributos, la pieza de jugador uno y dos. Si el tablero se borra, las fichas se borran con el también. Board instancia las piezas a través del atributo color de Player, por lo que se puede decir que depende de este objeto, ya que lo utiliza de forma temporal.

La clase Game implementa la interface Tda_Game, Game tiene una agregación con Player, ya que en su constructor se le deben dar dos jugadores para almacenarlos como atributo, sin embargo, Player es definido afuera de Game, por lo que su vida no depende de este. Por el otro lado, hace composición con Board, ya que este se instancia dentro de Game para almacenarlo como atributo, y como Board se compone con Piece, Game tiene una dependencia con Piece, ya que se usan sus métodos de forma temporal para Board dentro de Game.

Todos estos objetos se instancian dentro del Main, en el cual se imple-

mento un menú interactivo para que el usuario pueda jugar.

6. Aspectos de implementación

La totalidad del proyecto fue hecho en el programa IntelliJ IDEA versión 2024.3 con el compilador Gradle y la única librería utilizada fue la predeterminada de Java con el SDK versión 11 en Windows 10. El proyecto se estructura en nueve archivos principales, los cuales contienen el Main con el menú interactivo, las interfaces TDA Player, Piece, Board, Game y sus respectivas clases que las implementan.

7. Instrucciones de uso

La entrega fue compilada en el entorno de IntelliJ IDEA 2024.3 con Gradle, SDK Java 11 y en Windows 10.

La entrega cuenta con una carpeta con la documentación con JavaDoc, para acceder a la pagina con la información, se debe abrir el archivo con nombre: "index.html" el cual deberá direccionarlo a la siguiente pagina [3].

Para poder jugar, se debe abrir la carpeta de código fuente, donde se encontraran dos carpetas: .idea y lab3_PP, hacer click derecho en la dirección de la carpeta código fuente y copiarla [4].

Luego se debe abrir IntelliJ IDEA, seleccionar arriba a la izquierda la opción FILE y después la opción de Open [5]. Seguido a eso se debe pegar la direccion y clickear Ok [6] y presionar el boton de run en la parte superior a la derecha [7], inmediatamente se debe haber cargado la consola, con mensajes de que el codigo se esta compilando con Gradle, seguido por el menu principal del juego[8].

Posibles errores:

Si no tiene el SDK de Java 11, el programa le avisará y le pedira descargar una version de SDK, usted debe elegir cualquiera de las opciones de las versiones 11 brindadas por IntelliJ.

8. Resultados y autoevaluación

Se obtuvo un proyecto exitoso y funcional en su totalidad, se realizaron pruebas para todo tipo de caso, como posicionar una ficha en una columna fuera del rango o en una columna llena, que si es el caso, retorna el juego sin cambios. También se vieron los casos de victoria vertical, horizontal y

diagonal, asimismo con el caso de tener un empate. Todas estas pruebas se hicieron manualmente probando los diferentes casos extremos.

A continuación se muestran ejemplos de creación de juego [9], mensaje de victoria después de realizar una jugada, seguido de un mensaje por si se quiere seguir jugando [10], seguido de la opción de mostrar estadísticas [11] y la de ver el historial de la partida actual [12].

El programa obliga a elegir entre rojo o amarillo y entre 4 a 21 fichas al crear el juego. Cuando un juego termina, permite elegir si seguir jugando o no, si escoje si, el tablero se vaciará, el historial se restablecerá y las fichas serán devueltas a los jugadores y se debe seguir jugando normalmente, si escoje no, el juego actual se da por terminado, se podrán seguir viendo las estadísticas de ese juego, pero no se podrá seguir jugando, si se quiere seguir jugando después de elegir no, se debe crear un nuevo juego con nuevos jugadores. Finalmente, si se desea cerrar el juego se debe seleccionar seis para terminar el programa.

9. Conclusiones del trabajo

Para concluir, se puede decir que la dificultad del proyecto estuvo en el diseño que debería tener el programa, uno se pudo ayudar del UML para organizarse mejor. El proceso de implementación fue mucho mas sencillo respecto a los otros dos laboratorios, ya que este lenguaje tiene aspectos imperativos que ya se conocían.

El tiempo de realización fue mucho más corto con respecto a los laboratorios anteriores. En ellos se debió pensar más en como implementar las funciones, pero en este se tuvo que pensar más el diseño. Una limitación que se notó fue la de utilizar una ventana y poder jugar con imágenes en vez de texto. Hubiese sido épico poder crear un juego parecido a los que se pueden instalar en Steam.

Gracias a la constancia, y al tipo de trabajo, el cual es cercano a la vida laboral, se logró un proyecto exitoso y se adquirió un nuevo conocimiento, el cual puede ser muy útil para el futuro laboral y académico.

10. Referencias

Gonzalo Martinez – Paradigma Orientado a Objetos. (2024). `https://docs.google.com/presentation/d/1BZG0NFo6pybbpj9_tKQAignNHZKBNa8xrDqUHy_Fu84/edit#slide=id.g31ad8730be4_0_1962`

Gonzalo Martinez – Técnicas OOP: Tipo de Relaciones. (2024). https://docs.google.com/document/d/1UVxvSmXh-ORJP3faLPQuhzcwOezH9S5b7uAbFVGqX
edit?tab=t.0#heading=h.wyroc3yk99g1

Gonzalo Martinez – Técnicas OOP: Tecnicas Polimorfismo. (2024). https://docs.google.com/document/d/1SQ7aBaEGLqe4uJBA0YRmbJeLUJd1ZXv3OyHn1spui
edit?tab=t.0#heading=h.wyroc3yk99g1

11. Anexos

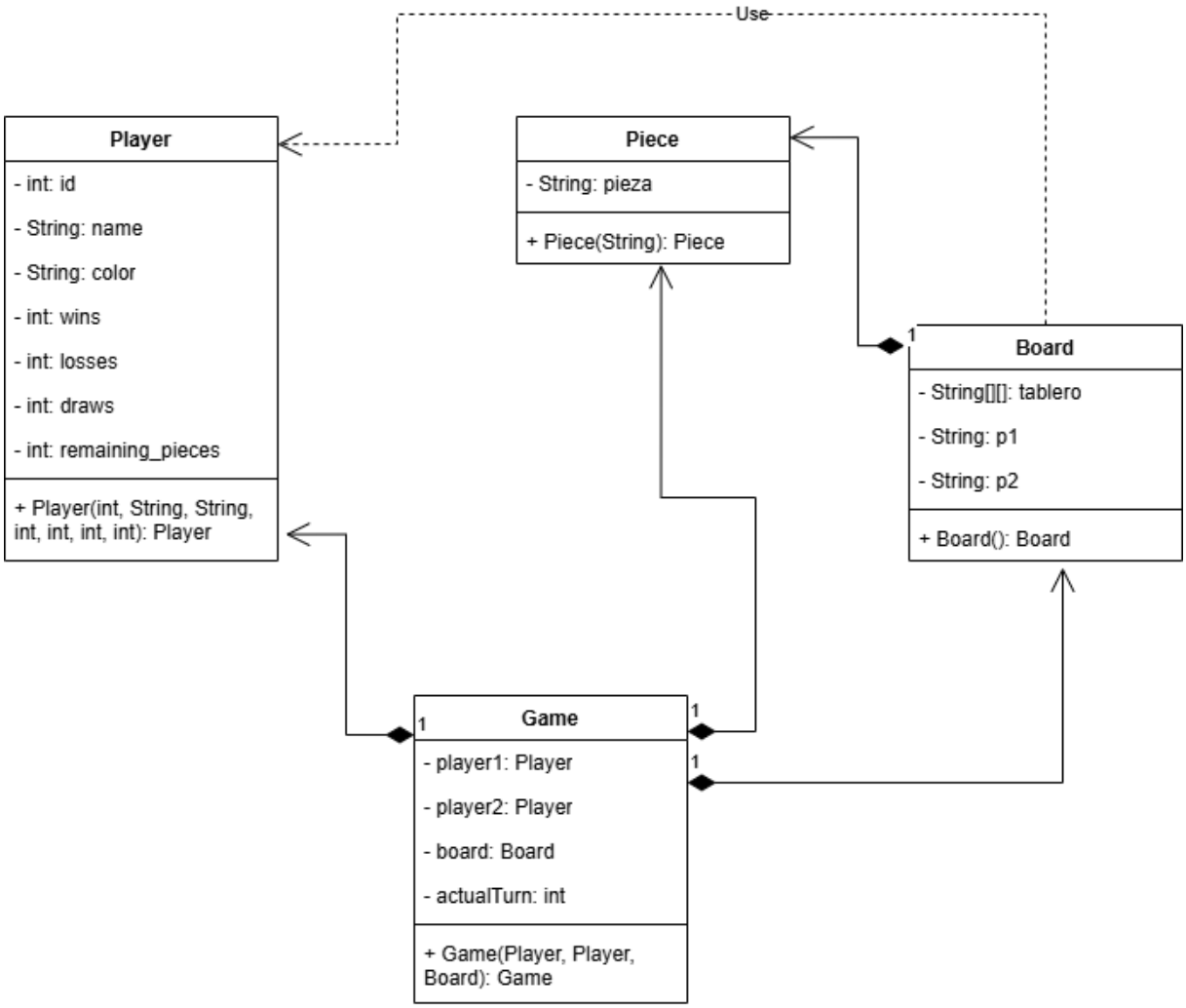


Figura 1: Diagrama de clases antes del desarrollo de software

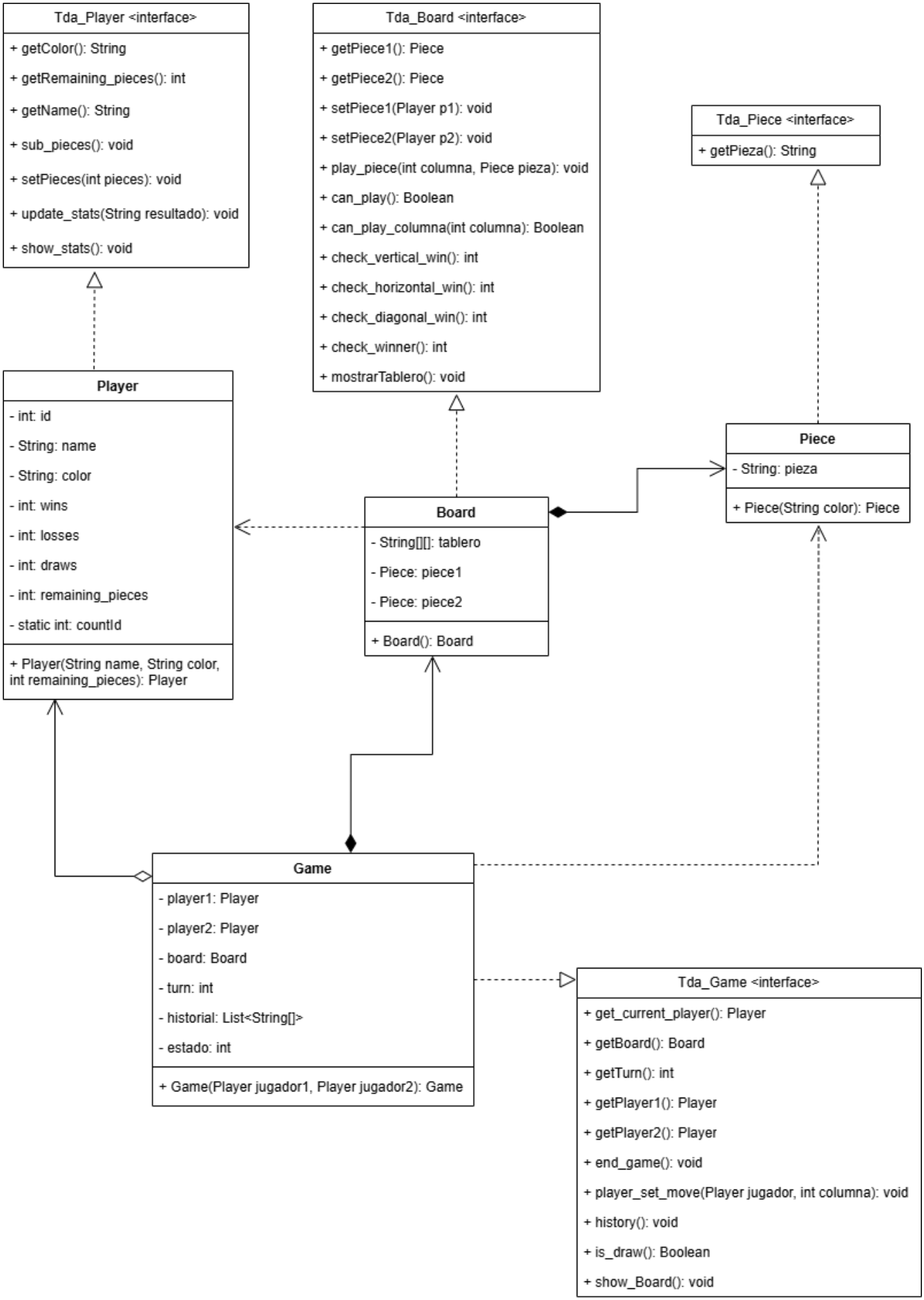


Figura 2: Diagrama de clases después del desarrollo de software

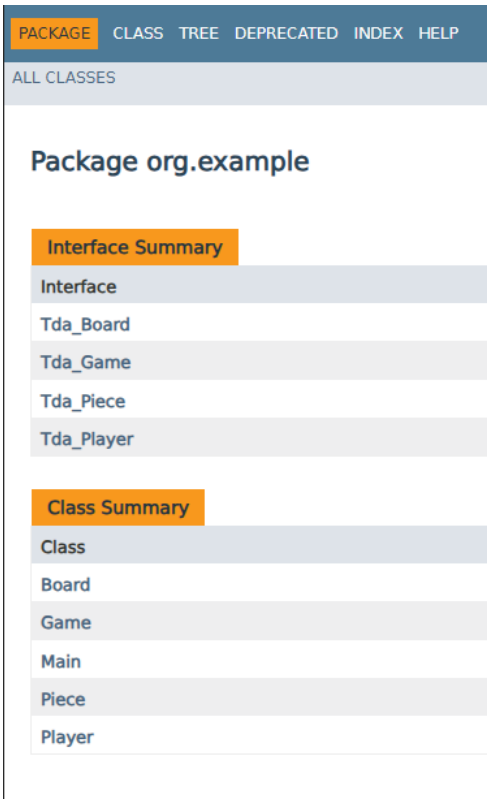


Figura 3: Pagina de JavaDoc

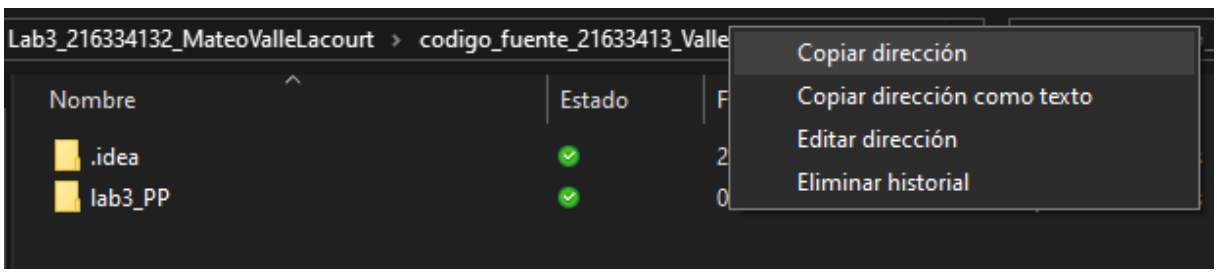


Figura 4: Imagen de referencia para copiar la dirección

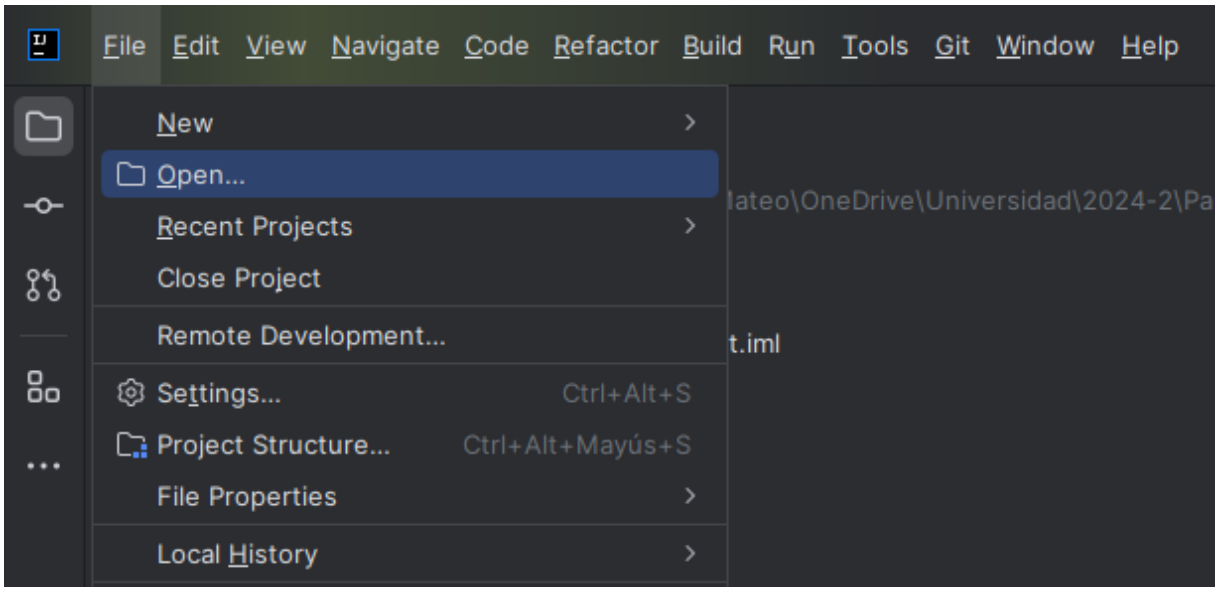


Figura 5: Imagen de referencia de IntelliJ IDEA para abrir proyecto

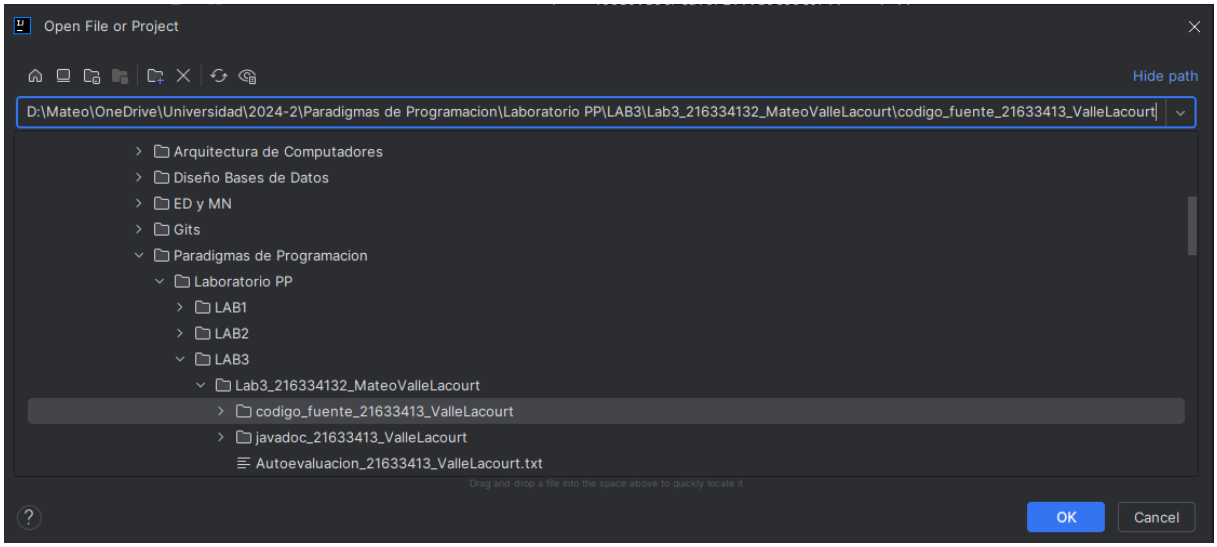


Figura 6: Imagen de referencia de IntelliJ IDEA para abrir proyecto

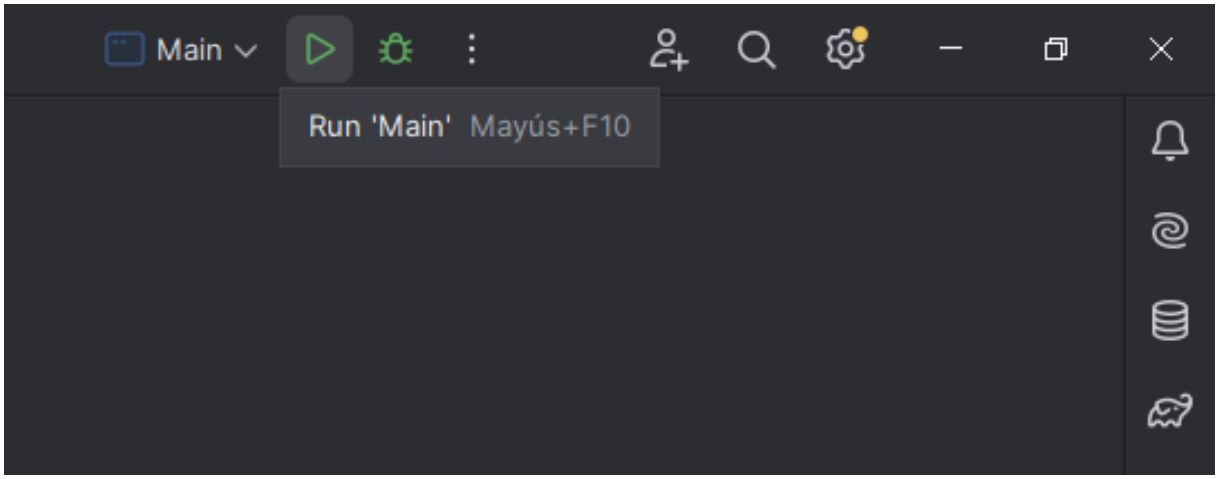


Figura 7: Botón run para arrancar el programa

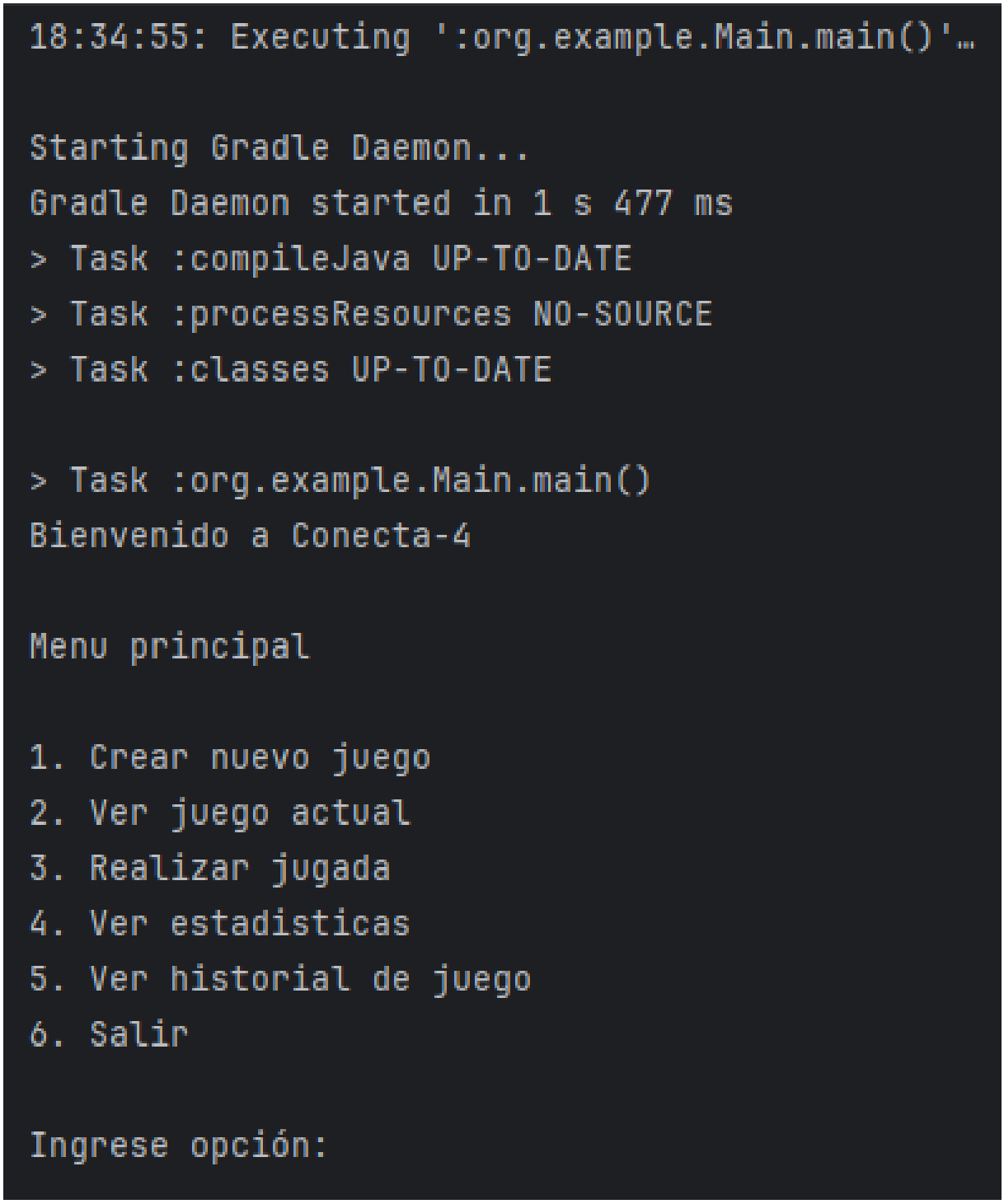


Figura 8: Imagen de el programa siendo compilado con Gradle y luego el menú cargado

```
-- Configuracion Jugador 1 --
Ingrese nombre del jugador 1: Mateo
Ingrese el color del jugador 1(Rojo/Amarillo): rojo

-- Configuracion Jugador 2 --
Ingrese nombre del jugador 2: Ivana
Ingrese el color del jugador 2(Rojo/Amarillo): Amarillo

-- Configuración de juego --
Ingrese la cantidad de fichas (4-21): 21
```

Figura 9: Ejemplo de creación de un juego

```
Ingrese opción: 3
### Realizar Jugada ###
Turno de Mateo(ROJO)
Cantidad de fichas: 16

Seleccione columna (1-7):
4
### Movimiento realizado: ###
| ( ) ( ) ( ) ( ) ( ) ( ) ( ) |
| ( ) ( ) ( ) ( ) ( ) ( ) ( ) |
| ( ) ( ) ( ) (R) ( ) ( ) ( ) |
| ( ) ( ) (R) (A) ( ) ( ) ( ) |
| ( ) (R) (A) (A) ( ) ( ) ( ) |
| (R) (A) (A) (R) ( ) ( ) (R) |
|-----------------------------|
### Mateo GANA!!! ###
### Estadísticas Actualizadas ###
Mateo(ROJO)
Victorias: 1
Derrotas: 0
Empates: 0

Ivana(AMARILLO)
Victorias: 0
Derrotas: 1
Empates: 0
-- ¿Quieren seguir jugando? --
Ingrese Y/N:
```

Figura 10: Ejemplo de jugada con victoria

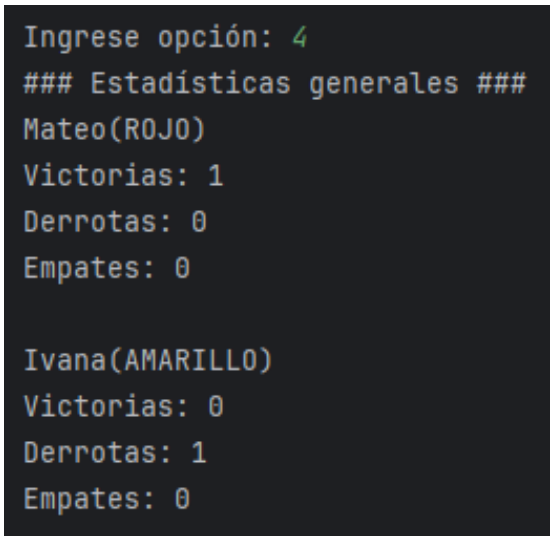


Figura 11: Opción ver estadísticas

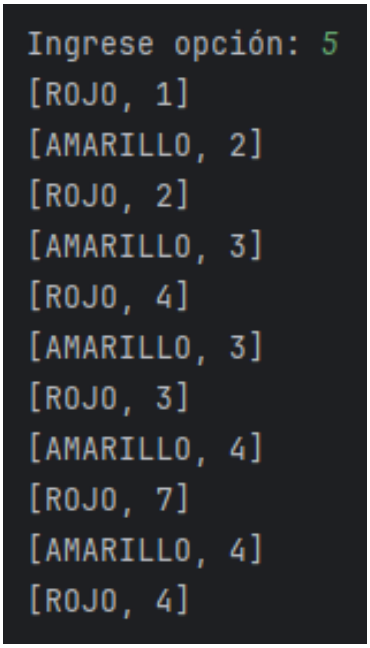


Figura 12: Opción ver historial