
# INSTITUTO TECNOLÓGICO DE COSTA RICA

## PROGRAMACIÓN ORIENTADA A OBJETOS


*Por:*
*Danilo Duque Gómez*
*Pablo Pérez Delgado*
*Jose Emmanuel Rojas Soto*

Profesor:
Andrés Víquez Víquez

21 de octubre de 2023

# 1. Introducción

En el curso de Programación Orientada Objetos, se tuvo como objetivo completar la codificación completa del famoso juego de mesa Rummikub. Este juego de mesa combina elementos de suerte, habilidad estrategia con números y colores. El juego utiliza fichas numeradas y se juega en rondas donde los jugadores intentan deshacerse de todas sus fichas formando combinaciones válidas en la mesa. Estas combinaciones pueden ser grupos de números del mismo valor pero de colores diferentes o secuencias numéricas del mismo color.


El juego fue inventado por Ephraim Hertzano, un ciudadano rumano-israelí, en los años 30. Originalmente, el juego se llamaba Rummikub en honor a la familia de Hertzano, y la palabra Rummikub.[es] una combinación de los nombres de sus hijos, Rhoda, Eliezer, Menachem y Yehuda.

La solución implementada fue programada en base al paradigma orientado a objetos. La programación orientada a objetos se basa en la idea de encapsular datos y funciones en objetos que interactúan entre sí a través de mensajes. Esto permite una mayor modularidad y reutilización de código. Además, la programación orientada a objetos facilita la organización y estructuración del código, lo que resulta en un desarrollo más eficiente y mantenible.

¿Pero por qué se utiliza este paradigma específicamente para la resolución del proyecto?

# 2. Estrategia de la solución

La solución implementada se desarrolló siguiendo el paradigma orientado a objetos. Esta elección se fundamenta en su capacidad para representar objetos del mundo real, específicamente el juego Rummikub. Se optó por este enfoque debido a su habilidad para modelar de manera efectiva las cualidades y características del juego, facilitando así la resolución lógica del programa.

Al utilizar objetos que representan las diversas partes del juego en la realidad, se simplifica significativamente la implementación, ya que cada componente del juego se puede abordar como una entidad independiente. Esto no solo mejora la estructura y organización del código, sino que también facilita el mantenimiento y la escalabilidad del sistema a medida que se introducen nuevas funcionalidades o se realizan modificaciones en el futuro.

Una estrategia adoptada que viene de la mano con lo anteriormente mencionado fue la distribución del trabajo en el equipo. Al ser un equipo de tres personas, la carga de trabajo se distribuyó de la siguiente manera: un encargado específicamente del diseño del software y gestión del equipo, un encargado en la lógica del programa y por último un encargado en la interfaz de usuario. Esta distribución permitió que cada miembro del equipo se especializara en su área de expertise, maximizando así la eficiencia y calidad del trabajo realizado. Además, esta estrategia también facilitó la comunicación y colaboración entre los miembros del equipo, ya que cada uno tenía claro cuál era su responsabilidad y cómo contribuía al objetivo general del proyecto.

Por último, se usó controladores de versión como github para facilitar la gestión y seguimiento de los cambios realizados en el código, lo que permitió un mejor control y organización del desarrollo del programa. Además, el uso de controladores de versión también facilitó la colaboración entre los miembros del equipo, ya que les permitió trabajar de forma simultánea en diferentes partes del proyecto sin interferir con el trabajo de los demás. Además, la implementación de controladores de versión como github también brindó la posibilidad de realizar rollbacks en caso de algún error o problema grave en el código, garantizando así la estabilidad del proyecto. De esta manera, el equipo pudo trabajar de manera más segura y confiable, evitando la pérdida de trabajo debido a errores que pudieran surgir durante el desarrollo.

# 3. Detalles de implementación

La fase de diseño, programación e implementación consto de dos partes importantes:

## 3.1. Implementación de la Lógica

Para llevar a cabo la implementación de la solución diseñada, se optó por modularizar al máximo las diferentes partes del juego. Esto implicó descomponer el juego en sus elementos más básicos y autónomos, convirtiéndolos en objetos individuales. La primera clase que se desarrolló fue la Clase Ficha. Este objeto de la vida real, de manera abstracta, se puede representar a través de tres atributos principales: Número, Color y un valor booleano que indica si la ficha es un joker o no.


Con la implementación de la clase que representa una ficha en el juego, se procedió a crear la clase encargada de gestionar un almacén de fichas, denominada Almacén. Esta clase tiene un único atributo, un vector que almacena las 106 fichas del juego. Sin embargo, su función va más allá de la simple conservación de fichas; también se encarga de barajarlas y distribuir 14 fichas a cada jugador. Además de estas tareas fundamentales, la clase Almacén ofrece métodos que permiten obtener información detallada sobre las fichas almacenadas, como el número total de fichas o la cantidad restante después de haber sido distribuidas. Estos recursos

no solo facilitan un control eficaz del desarrollo del juego, sino que también simplifican la implementación de reglas adicionales, proporcionando una base sólida para la dinámica del juego.

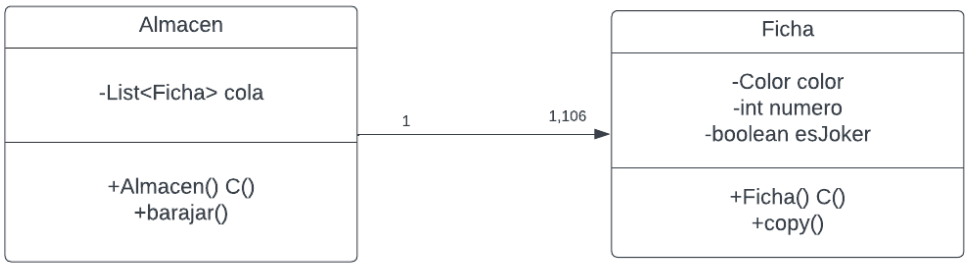



Figura 1: El Anterior diagrama representa la interacción de las clases en cuestión.

Contando con las clases de Almacen y Fichas, ahora el siguiente paso es empezar a idear una clase que de manera abstracta represente a un jugador, y mediante el estudio de los atributos necesarios de un jugador, se concluyo que era indispensable trabajar el mazo de un jugador como un objeto de la clase Baraja. Esta clase permitirá al jugador tener un conjunto de cartas con las que podrá interactuar durante el juego. Además, esta clase también facilitará la implementación de reglas relacionadas con el manejo y la gestión de las cartas en el juego. Por último, se previo que el único atributo necesario para representar un mazo de cartas era un vector de fichas que, al inicio del juego, se llenaba con 14 cartas elegidas al azar (previamente mezcladas por el almacen).

Con esta implementación, se ha logrado la completitud de la clase Baraja. En resumen, dicha clase se encarga de tomar 14 fichas del almacén al ser creada, así como de tomar fichas del almacen en caso de que el jugador en cuestión no disponga de ninguna jugada legal.

Con la incorporación de un nuevo atributo diseñado para representar la baraja del jugador, se procedió a desarrollar la clase del jugador mediante un cuidadoso diseño y programación. Durante este proceso, se consideró esencial incluir otros dos atributos esenciales para representar completamente al jugador: su nombre y la cantidad de puntos que ha acumulado desde el inicio de la ronda de partidas.

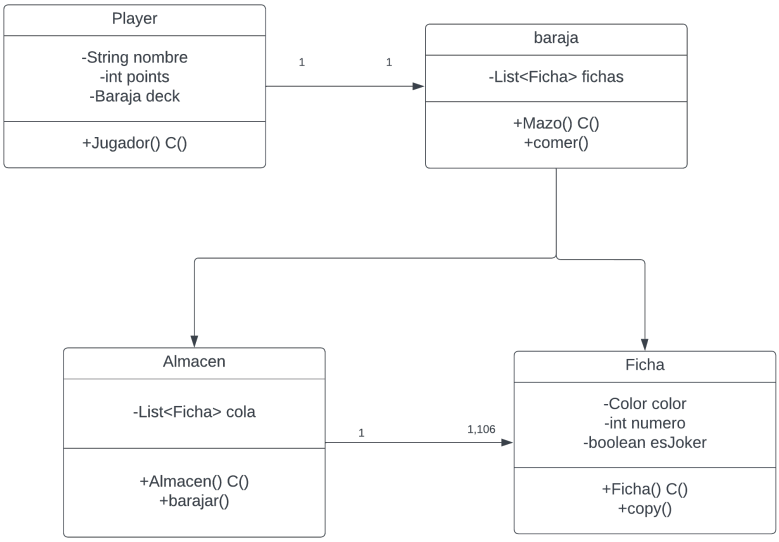



Figura 2: El Anterior diagrama representa la interacción de las nuevas clases introducidas.

Una vez implementada la lógica interna de los jugadores y su funcionamiento, el siguiente paso consiste en diseñar la parte del programa encargada de gestionar el tablero, así como de verificar la validez de cada jugada.

Tras un exhaustivo análisis sobre los atributos necesarios para representar un tablero, se llegó a la conclusión de que este sería notablemente similar a los tableros de ajedrez o damas chinas. En este contexto, los jugadores solo pueden colocar fichas o piezas en posiciones específicas, lo que implica que las jugadas válidas se limitan a movimientos horizontales, es decir, a lo largo de las filas del tablero.

Para simplificar y modularizar la estructura del tablero en el juego Rummikub, se determinó que era esencial diseñar un objeto que representara una fila. Este objeto no solo tendría la función de visualizar la disposición de las fichas, sino también de verificar la validez de las jugadas efectuadas en esa fila específica.

Esta clase se denomina Casilla y posee un único atributo: un arreglo estático de Fichas con un tamaño de 20. Se han implementado diversos métodos para esta clase, incluyendo la capacidad de verificar tanto Series como Grupos, así como

añadir y remover fichas de manera eficiente.

Gracias a esta estructura, la creación del tablero se simplificó enormemente. Todas las verificaciones necesarias se llevan a cabo en el objeto Casilla específico que está involucrado en la acción que el usuario desea realizar. Por ende, el tablero se define como un arreglo estático compuesto por siete casillas. Cuando se evalúa la validez de una jugada realizada por el usuario, lo que realmente sucede en el nivel técnico es la aplicación de los métodos implementados en la clase Casilla para cada una de las siete casillas del tablero.

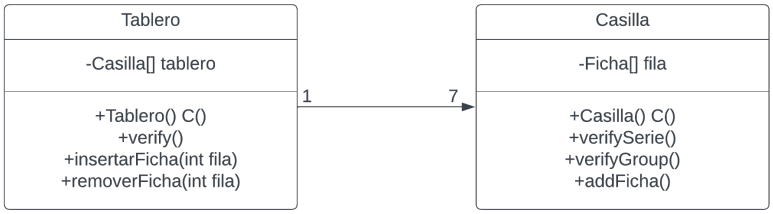



Figura 3: El Anterior diagrama representa la interacción de la clase Tablero con la clase Casilla.

Con todas las clases previamente mencionadas, se ha logrado modelar el juego Rummikub de manera completa. Sin embargo, aún queda una última pieza fundamental: una clase que implemente todas las funcionalidades de manera coherente, permitiendo así que el usuario pueda aprovechar todas las características del juego. A esta clase se le ha denominado "Game". La clase Game recibe como parámetro la cantidad de jugadores que participarán en la partida y se encarga de gestionar de manera eficiente a qué jugador le corresponde jugar en cada turno. Además, se encarga de mantener el estado actual del tablero y el almacen de fichas en todo momento, permitiendo que cada jugador, cuando sea su turno, pueda realizar las jugadas que prefiera de manera fluida y sin complicaciones.

Se consideró esencial la implementación de un vector de objetos Player para manejar el flujo de jugadores con sus respectivos turnos en la clase Game. Así como un tablero que lleve guardadas todas la fichas colocadas en él de manera legal, un almacen de fichas del que los jugadores puedan tomar fichas en caso de no poder hacer una jugada legal, y por último una estructura de datos que guarda

los puntajes de los jugadores en el momento que la partida acabe.

El manejo de los datos de los jugadores se ha optimizado con la implementación de la clase Archive. Esta clase utiliza un HashMap, donde se almacenan los puntos de cada jugador asociados a sus respectivos nombres. La elección de un HashMap para esta funcionalidad resulta altamente beneficiosa debido a su eficiencia tanto en la inserción como en la modificación de elementos. Además, esta estructura de datos permite mantener registros de todas las personas que han participado en las partidas hasta el momento, incluso si son diferentes personas en comparación con la partida anterior.

Esta flexibilidad en el manejo de datos del usuario se traduce en una experiencia más dinámica y adaptable, permitiendo conservar información precisa sobre los jugadores a lo largo del tiempo. En conjunto estos cuatro atributos representaban a la clase Game de manera abstracta, logrando toda la funcionalidad esperada de esta clase.

Con la implementación de estas clases, se ha completado la totalidad de la lógica del juego, permitiendo tener una partida funcional (aunque sin la parte gráfica) que sigue las reglas establecidas en Rummikub. Además, se ha logrado la capacidad de almacenar los datos ingresados por cada usuario para su posterior uso en la tabla de datos. Sin embargo, todavía queda pendiente la parte visible para el usuario: la interfaz gráfica.

## 3.2. Implementación de interfaz gráfica

Luego de diseñar la logística del juego, se procedió a implementar los componentes gráficos. La herramienta principal utilizada para crear esta interfaz fue Java Swing, una de las muchas bibliotecas incluidas en Java. Java Swing facilitó en gran medida la creación de la interfaz gráfica, gracias a la capacidad de herencia que ofrece la programación orientada a objetos. Esto permitió aprovechar las clases ya existentes e incorporarles componentes gráficos.
Las clases que se beneficiaron de Java Swing en esta implementación fueron las siguientes: la clase Ficha, la clase Player y las clases Game, MainMenu, ResultGUI y EndScreen. A continuación, se explicará cómo se implementaron estos componentes gráficos.

Ficha

La clase Ficha extiende la clase JButton, utilizándose JButton porque las fichas necesitan ser interactivas para poder moverlas en el tablero. JButton permite mostrar las fichas en la pantalla del jugador y agregarles los datos específicos de la clase Ficha de manera sencilla.
El comportamiento del botón de la ficha varía según el tipo de ficha que represente. Puede ser una ficha en el mazo del jugador o una en el tablero. En el tablero, puede ser una ficha vacía o una ficha con elementos en ella.
Se utilizan fichas vacías en el tablero para facilitar la adición de fichas desde el mazo al tablero. Este proceso se logra mediante una variable llamada "buffer"que, en el momento en que se presiona una ficha en el mazo, copia todos los datos de esa ficha. Cuando el jugador decide dónde colocar la ficha, la ficha vacía en el tablero copia los datos del "buffer". Luego de esto, "buffer"se establece como nulo para evitar la copia repetida de la misma ficha.
Player

La clase Player extiende la clase JFrame, y la razón para esto es que por cada instancia de jugador que se crea, el juego abre una nueva ventana utilizando JFrame para solicitar el nombre del jugador. Esta ventana incluye un JButton que se utiliza para guardar los datos que el jugador ingresa, un JTextField para almacenar los datos del jugador y un JLabel que muestra un icono del jugador para informar al usuario sobre lo que está ocurriendo. Esta clase implementa ActionListener para poder usar los botones.

Game, MainMenu, ResultGUI y EndScreen

Las clases Game, MainMenu y ResultGUI extienden JFrame. Todas estas clases tienen funcionalidades muy similares, ya que sirven como plantillas que permiten agregar clases previamente creadas, como Ficha y Baraja. Game es la clase principal y la que la mayoría del tiempo será visible para los jugadores. Se encarga de mostrar el tablero y la baraja de cada jugador, además de proporcionar botones necesarios como el botón de çomer", reiniciar tableroz realizar jugada". En esta clase, también se agregó un Tablero que se encarga de mostrar todas las fichas vacías.
Para la implementación de la baraja, se optó por utilizar un arreglo estático que

copia los datos de la baraja del jugador al que pertenece el turno. Este enfoque ahorra memoria y simplifica considerablemente el proceso de actualizar la baraja.

MainMenu es la clase que se mostrará inmediatamente después de ejecutar el programa. Esta clase se encargará únicamente de mostrar el nombre del juego con un JLabel que contiene el logo de Rummikub. También se encargará de solicitar al usuario la cantidad de jugadores que desean participar. Para lograr esto, se utiliza un JTextField con una validación para asegurarse de que el jugador ingrese siempre un número entre 1 y 4. Si el usuario no ingresa un número válido, se mostrará un mensaje de error utilizando JOptionPane. MainMenu implementa la interfaz ActionListener para dar funcionalidad al botón de confirmación que se encuentra en la pantalla y se utiliza para confirmar la cantidad de jugadores.

ResultGUI se encarga de mostrar los puntajes de los jugadores al final de la partida. Lo hace utilizando dos JLabels, uno para mostrar el nombre del jugador y otro para mostrar la cantidad de puntos del jugador. Para que ResultGUI tenga acceso a estos datos, se le pasa un archivo como parámetro que almacena la información de todos los jugadores.
Además, esta clase implementa la interfaz ActionListener, ya que utiliza dos botones. Uno de los botones se utiliza para iniciar una nueva partida, mientras que el otro se utiliza para finalizar el juego y mostrar la pantalla EndScreen.

EndScreen es una clase que se utiliza al final de la ejecución del programa. Esta clase se encarga de mostrar el nombre del jugador junto con una imagen de felicitación por haber ganado. Esto se logra mediante el uso de dos JLabels. Es importante mencionar que esta clase no incluye botones, ya que en este punto del programa, la ejecución ha finalizado, y el jugador puede usar el botón de salida para cerrar la aplicación.

# 4. Restricciones y suposiciones

Al desarrollar el proyecto del juego Rummikub, se han tenido en cuenta ciertas restricciones y suposiciones que han guiado el proceso de diseño e implementación. Estas son las restricciones y suposiciones clave consideradas durante el desarrollo del proyecto:

## 4.1. Restricciones

- **Plataforma de Desarrollo:** El juego Rummikub se ha desarrollado utilizando Java y la biblioteca Swing para la interfaz gráfica. Esto limita la ejecución del juego a plataformas que admiten Java.

- **Cantidad de Jugadores:** El juego ha sido diseñado para admitir de 1 a 4 jugadores. Aun que las funcionalidades implementadas si funcionan en caso de más jugadores, el almacen de fichas esta creado con 106 fichas, por lo que para repartir 14 fichas a cada jugador, se necesitan que sean menos de 6 usuarios.

- **Interacción Gráfica:** La interacción de los jugadores con el juego se realiza a través de la interfaz gráfica. No se ha implementado una versión de línea de comandos o una interfaz de usuario basada en texto.

- **Reglas del Juego:** El juego sigue las reglas estándar del Rummikub. No se han implementado variaciones o reglas personalizadas.

- **Idioma:** La interfaz gráfica y las instrucciones del juego están en español. No se han implementado funcionalidades de cambio de idioma.

- **Sistema Operativo:** El juego se ha desarrollado y probado en sistemas operativos compatibles con Java. Las diferencias en el comportamiento del sistema operativo pueden afectar la experiencia del usuario

- **Resolución de la pantalla:** La resolución de la pantalla sobre la que se ve el programa es importante a la hora de que todo esta alineado de la manera en la que se propuso, para que el programa, se vea bien y se recomendó una pantalla de 2560x1660 píxeles.

## 4.2. Suposiciones

- **Conocimiento del Juego:** Se supone que los jugadores tienen conocimiento de las reglas y el funcionamiento del juego Rummikub. No se proporcionan tutoriales detallados sobre cómo jugar.

- **Integridad de Datos:** Se asume que los datos proporcionados por los jugadores, como sus nombres, son válidos y únicos por lo que dos jugadores no podrán llamarse igual.

- **Condiciones de Ejecución:** Se asume que el entorno de ejecución del juego (como la máquina virtual de Java y las bibliotecas utilizadas) funcionará correctamente y no habrá problemas inesperados durante la ejecución.

- **Condiciones de Empate:** En caso de empate, se asume que no se necesitan reglas adicionales para determinar el ganador y que los jugadores están de acuerdo en compartir la victoria en caso de puntuaciones iguales.

- **Disponibilidad de Recursos del Sistema:** Se supone que la computadora o dispositivo en el que se ejecutará el juego tiene suficientes recursos del sistema (como memoria RAM y capacidad de procesamiento) para garantizar un rendimiento fluido y sin problemas.

# 5. Problemas encontrados

Durante el diseño e implementación del juego Rummikub, se encontraron varios problemáticas que era necesario perfeccionar antes de poder llegar a la completitud de lo esperado, algunos de los problemas encontrados durante el desarrollo del juego Rummikub incluyeron:

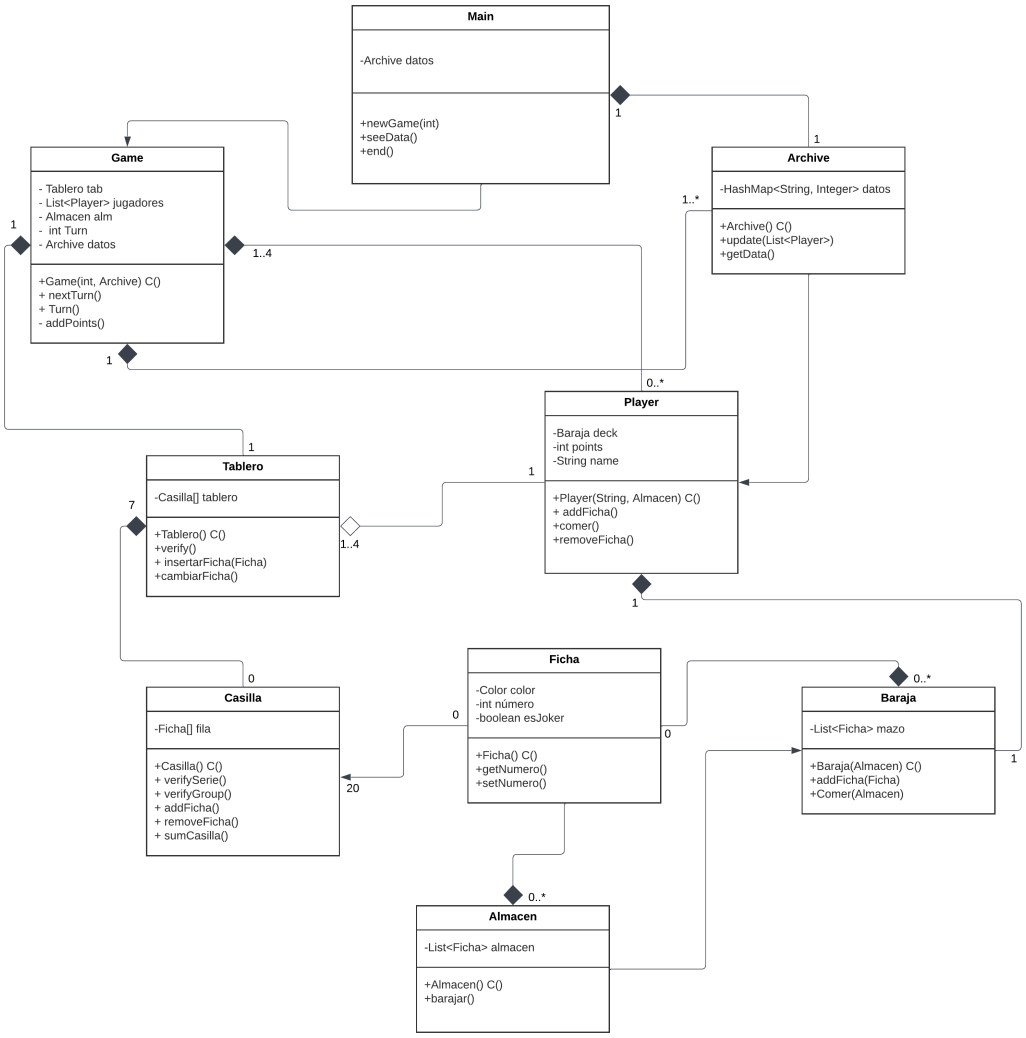
- **Gestión del Estado del Juego:** Se implementó una estructura de datos sólida y métodos eficientes para mantener un seguimiento preciso del estado del juego, incluyendo los turnos de los jugadores, el estado del tablero y las fichas disponibles en el almacén. Esta tarea fue esencial para garantizar un flujo de juego suave.

- **Validación de jugadas:** Se desarrollaron algoritmos detallados para verificar la validez de las combinaciones de fichas en el tablero, así como las jugadas realizadas por los jugadores. Esto implicó la creación de métodos sofisticados para verificar grupos y secuencias, asegurándose de que las jugadas cumplían con las reglas de Rummikub.

- **Coordinación del Equipo:** Dado que el proyecto implicaba trabajo en equipo, se coordinaron las tareas y se aseguró una comunicación efectiva entre los miembros del equipo. Se mantuvieron sincronizados en cuanto a las implementaciones y se resolvieron problemas de integración cuando fue necesario.

- **Gestión en errores de usuario:** En el transcurso del juego, es común que los usuarios cometan errores, como intentar realizar movimientos inválidos o ingresar datos incorrectos. Implementar mensajes de error que sean claros y comprensibles se convirtió en una tarea fundamental. Esta implementación cuidadosa es esencial para orientar a los usuarios, y hacer que la ejecución se haga más llevadera.

# 6. Diagrama de clases



# 7. Conclusiones y recomendaciones

En el transcurso del proyecto de desarrollo del juego Rummikub, el equipo de programadores logró alcanzar una serie de logros significativos. A través de la aplicación práctica de los principios de la programación orientada a objetos, se pudo diseñar e implementar una solución completa para el juego de mesa, que abarca desde la lógica del juego hasta la interacción de los usuarios a través de una interfaz gráfica intuitiva.

Durante este proceso, el equipo se enfrentó a diversos desafíos, incluyendo la gestión del estado del juego, la validación de jugadas y la coordinación eficiente entre las clases del sistema. Estos desafíos no solo permitieron mejorar las habilidades técnicas del equipo, sino que también fomentaron la colaboración y el trabajo en equipo, esenciales en el ámbito de la programación.

Uno de los puntos clave de mejora identificados se encuentra en la interfaz gráfica del juego. Aunque funcional, se recomienda dedicar esfuerzos adicionales para mejorar la experiencia del usuario. Esto incluye implementar animaciones suaves, indicadores visuales claros para las jugadas válidas y una disposición estéticamente agradable del tablero y las fichas. Estas mejoras no solo incrementarían la usabilidad del juego, sino que también añadirían un toque de profesionalismo a la experiencia del usuario.

Además, se destaca la importancia de optimizar el código para mejorar el rendimiento del juego. Esto implica revisar y refinar el algoritmo de validación de jugadas, así como gestionar eficientemente los recursos del sistema para asegurar una experiencia de juego fluida y sin interrupciones.

En futuras iteraciones del proyecto, se podría considerar la implementación de funcionalidades adicionales que enriquezcan la experiencia de juego. Por ejemplo, la adición de una inteligencia artificial que permita a los usuarios jugar contra la computadora o la introducción de un sistema de puntuación más complejo que tome en cuenta las estrategias de los jugadores. Estas adiciones podrían agregar profundidad y emoción al juego.

Además, se subraya la importancia de llevar a cabo pruebas exhaustivas del juego en diferentes escenarios y situaciones. Esto ayudaría a identificar posibles errores y a garantizar la estabilidad y robustez del juego. Además, mantener una

documentación detallada del código, incluyendo comentarios claros y explicativos, es esencial para facilitar la comprensión del sistema y permitir futuras actualizaciones y mantenimiento sin inconvenientes.

En resumen, el proyecto del juego Rummikub ha sido una valiosa experiencia que ha permitido al equipo aplicar conocimientos teóricos en un contexto práctico. Las lecciones aprendidas, los desafíos superados y las mejoras identificadas durante este proceso servirán como una sólida base para futuros proyectos, fortaleciendo así las habilidades y el conocimiento del equipo en el campo de la programación orientada a objetos.

# 8. Bibliografía


[1] Machuca, M. (2019). *Reglas del Rummikub*. *aboutespanol*. `https://www.aboutespanol.com/rummikub-2077603`

[2] Lesson 8: Object-Oriented Programming. (s. f.). Oracle. `https://www.oracle.com/java/technologies/oop.html`

[3] What is UML? The Unified Modeling Language (UML) is a graphical modeling language | SPARX Systems. (s. f.). SPARX Systems. `https://sparxsystems.com/platforms/uml.html`

[4] GitHub: Let's build from here. (s. f.). GitHub. `https://github.com/`