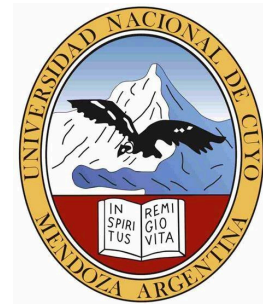


Paradigmas de programación

Informe laboratorio



Forni Diego, Massacesi
Juan Ignacio

Lic. Cs. de la
Computación/Facultad de
Ingeniería/UNCuyo

17 de septiembre de 2024

Abstract

Este trabajo se enfoca en el desarrollo del ejercicio integrador de la Unidad 2 de la materia Paradigmas de la Programación. El objetivo es modelar e implementar una versión del juego de mesa "Monopoly" utilizando el lenguaje de programación Java.

1. Introducción

En este informe, abordaremos el modelado, diagramado e implementación del juego de mesa Monopoly, siguiendo el paradigma de programación orientado a objetos. Este enfoque proporcionará una estructura clara y flexible para representar los diversos aspectos del juego, desde los componentes básicos hasta la interacción entre ellos.

Para esto, contamos con la siguiente lista de requerimientos:

1. Dado un tablero de monopoly, inicializar como mínimo con 2 jugadores y como máximo con 4. Se pide poder listar el nombre de todos los jugadores, tipo de pieza y su posición actual en el tablero.
2. Lograr que se den al menos 10 vueltas en el circuito e informar que pasa en cada tirada de dados.
3. Informar por consola o de manera visual como está el tablero.
4. Representar que un jugador pueda tirar los dados.
5. Representar la posibilidad de comprar un lugar o no.
6. Saber si un jugador esta en posibilidad de bancarrota o no .
7. Como implementaría una interfaz en el diagrama de clases. Agregar al diagrama de clases y en la implementación.
8. Debe existir al menos algún método polimórfico .

En cuanto la implementación, el objetivo es un juego ejecutable por consola con la posibilidad de juego automático. Para lograr buena compatibilidad, se utilizará Java 8.

Nuestro ideal es superar los requerimientos impuestos, pudiendo brindar una experiencia más similar a la del juego original. Para eso, hemos decidido incluir más aspectos del juego, tales como la cárcel, cartas de fortuna, etc.

2. Metodología

Decidimos al momento de realizar la abstracción de los objetos del juego tener en cuenta algunas consideraciones.

1. La clase Banco no pertenece a nuestra implementación, quien se encarga de suplir al banco es el propio juego. El juego se encargará de otorgar propiedades, realizar los pagos y cobros, lo cual simplifica la implementación del programa, además decidimos que los jugadores no sean banqueros para evitar trampas en la jugabilidad.
2. El dinero no lo representamos como billetes (siendo dos billetes de 500, dos de 100, dos de 50, seis de 20, cinco de 10, cinco de 5 y cinco de 1). Decidimos representarlo como una cantidad incorpórea que ya trae consigo el jugador. Eso nos da una mayor facilidad al manejo del dinero ya que nos evitamos que el jugador tenga que elegir con qué billete quiere pagar y cuantos quiere usar, además nos evitaremos problemas al devolver el cambio correspondiente a cada compra realizada.
3. El costo de las propiedades es invariante ya que decidimos no aumentar el costo de las propiedades de acuerdo al color comprado o la cantidad comprada. Con respecto a las propiedades también decidimos no implementar la compra de casas y hoteles para simplificar el manejo del dinero.
4. El manejo de hipotecas por parte del banco, lo reemplazamos por la decisión de eliminar al jugador de la partida una vez que este se encuentre en bancarrota. Un jugador se encuentra en bancarrota cuando no puede realizar un pago obligatorio, ej: rentas, impuestos, pagos(Cartas).
5. Las cartas de Arca y las cartas de Casualidad se implementaron en la misma clase ArcaOCasualidad, es decir, se simplificó la cantidad de casillas en una sola que contiene todas las cartas del juego. Donde las cartas se encuentran mezcladas en el mismo mazo.
6. El tablero está conformado por 20 casillas, por lo que los jugadores tendrán un solo dado para no avanzar tan rápido sobre el tablero.
7. Un jugador va a la cárcel cuando cae en esta o tiene una carta que se lo indica. No va la cárcel por sacar dobles y una vez en esta no tirara dobles (los jugadores tienen un solo dado).

2.1 Modelado

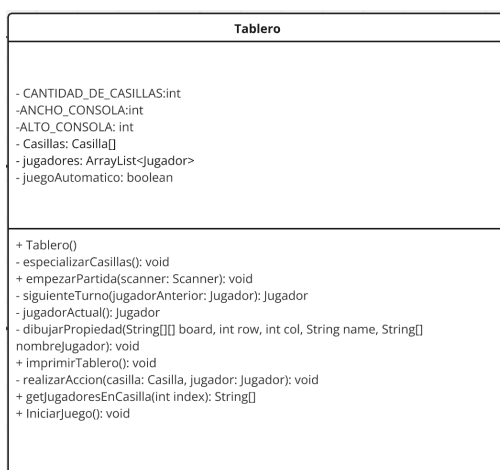
En esta sección se detalla el proceso de modelado del juego, nuestra abstracción del juego de mesa, desde los conceptos más básicos hasta la creación del diagrama final.

Durante este proceso, se identificaron y desarrollaron las siguientes clases principales, cada una con un papel crucial en la estructura del juego:

- **Tablero:**
 - Esta clase representa el tablero físico del juego y es responsable de coordinar todos los aspectos relacionados con él. Actúa como el núcleo central donde se colocan las casillas y se mueven los jugadores. El Tablero gestiona la disposición y la organización de las casillas, asegurando que el flujo del juego siga las reglas establecidas y que cada casilla funcione correctamente según su tipo y propósito.
- **Jugador:**
 - La clase Jugador se encarga de manejar las decisiones y acciones individuales de cada jugador en el juego. Almacena y gestiona la información esencial sobre el estado del jugador, como su posición en el tablero, el dinero disponible, y cualquier otra característica relevante que afecte su desempeño.
- **Pieza:**
 - Implementado como un enum, la clase Pieza permite a los jugadores seleccionar y usar una pieza específica en el tablero. El uso de un enum asegura que las piezas sean únicas y no se repitan durante el juego. Este enfoque facilita la gestión de las piezas y su asignación a los jugadores de manera organizada y eficiente.
- **Casilla:**
 - La clase Casilla es abstracta y sirve como base para una variedad de especializaciones, tales como Propiedad, Cárcel, Impuesto, y otras. Cada especialización de Casilla tiene atributos y comportamientos específicos que se adaptan a su función particular en el juego. Por ejemplo, una casilla de Propiedad tendrá atributos relacionados con el costo y la renta, mientras que una casilla de Cárcel gestionará las reglas de encarcelamiento.

Sin embargo, este conjunto inicial de clases representa solo una simplificación del modelo del juego. A medida que se avanzó en el desarrollo, surgieron nuevas necesidades y desafíos que llevaron a la creación de clases e interfaces adicionales. Algunas de estas nuevas clases surgieron para llenar vacíos en el sistema y abordar aspectos específicos del juego que no estaban contemplados en la primera fase. Otras fueron implementadas para mejorar la flexibilidad y robustez del juego, permitiendo una mayor personalización y adaptación a distintas reglas y escenarios. A continuación, se detallan estas adiciones y cómo contribuyen a enriquecer el diseño y funcionamiento del juego.

Tablero



La clase Tablero no solo representa el tablero tradicional del juego Monopoly, sino que es el encargado de la ejecución del juego, por esto lleva atributos como las dimensiones de la consola. Además de esto, posee relaciones de composición con Jugador y Casilla.

Cuenta con todos los métodos para inicializar y manejar el flujo del juego.

Otra simplificación que decidimos hacer para nuestro juego, es que el tablero, contiene solo 20

Figura 1. Clase Tablero.

casillas, a

diferencia de las 40 del juego original.

Jugador

Jugador
<ul style="list-style-type: none"> - posicion: int - pieza: Pieza - nombre: String - enBancarrota: boolean - dadosAnteriores: ArrayList<Integer> - propiedades: ArrayList<Propiedad> - dinero: int - turno: boolean - enCarcel: boolean - tieneCartaSalidaDeCarcel: boolean
Jugador(pieza: String) + tirarDado(): int + imprimirDadosAnteriores(): void + avanzar(): void + pagarRenta(renta: int) + recibirDinero(monto: int) + dineroRestante(): void + agregarPropiedad(propiedad: Propiedad): void + mostrarPropiedades(): void + eliminarPropiedad(): void

La clase Jugador representa a cada participante en la partida, almacenando información clave como el nombre, la pieza elegida, el dinero disponible y la posición en el tablero. Además de manejar estas propiedades, los jugadores son responsables de acciones cruciales como tirar los dados y tomar decisiones estratégicas durante el juego.

Figura 2. Clase Jugador.

Casilla

*Casilla
<ul style="list-style-type: none"> - nroDeCasilla: int - nombre: String - juegoAutomatico: boolean
+ Casilla(nombre: String) + Casilla(nombre: String, juegoAutomatico: boolean) + getNombre(): String *accion(jugador: Jugador): void

Hemos decidido modelar esta clase como una clase abstracta, ya que como tal, nunca será usada, sino que usaremos especializaciones de la misma.

Esto permite que cada tipo de casilla tenga un comportamiento específico, mientras que el uso del polimorfismo facilita que las interacciones con las casillas se realicen de manera uniforme en todo el juego. Por ejemplo, al caer en una casilla de tipo Propiedad, se activarán comportamientos distintos que si se cae en una casilla de tipo Impuesto o Arca. Esta estructura flexible y extensible es clave para manejar la complejidad del juego de Monopoly.

Figura 3. Clase abstracta casilla.

Mazo y Cartas

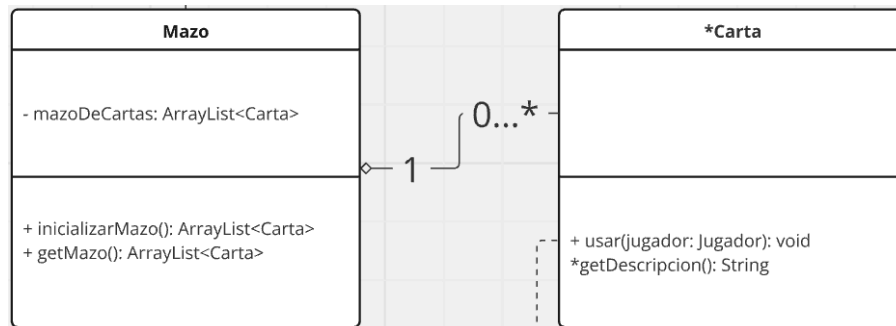


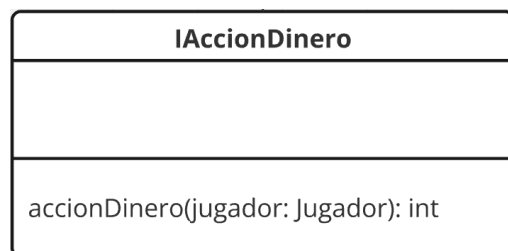
Figura 4. Clases Mazo y Carta.

Hemos modelado el mazo para que éste conste de múltiples cartas diferentes, brindando aleatoriedad y sorpresas al juego.

La clase carta es abstracta, ya que en el juego solo usaremos especializaciones de la misma.

Estas especializaciones son: CartaDinero, CartaSalirCarcel, CartaIrCarcel, y CartaAvanzarFerrocaril.

IAccionDinero



Durante el proceso de modelado del juego, se identificó que tanto las propiedades, como la casilla Adelante e Impuestos, realizan acciones sobre el dinero de un jugador. Para manejar esta funcionalidad de manera eficiente y evitar redundancia en el código, se decidió implementar una interfaz llamada IAccionDinero.

Figura 5. Interfaz IAccionDinero.

A continuación, mostramos el [diagrama UML completo realizado con](#)

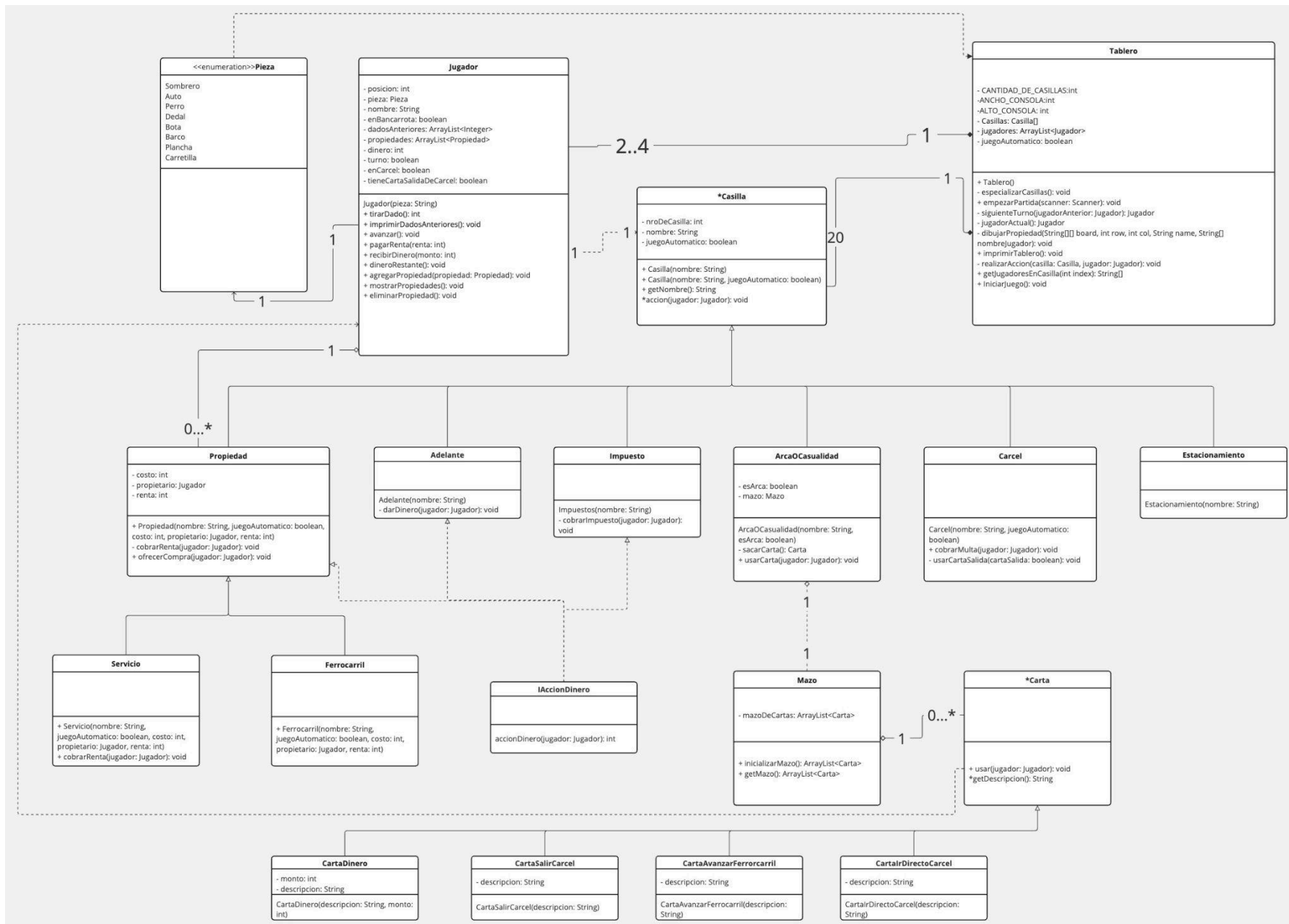


Figura 6. Diagrama UML.

2.2 Implementación

2.2.1 Configuración de equipos utilizados

Al momento de la implementación se utilizaron dos computadoras, una con distribución Windows 11 y la otra con Mac. Las versiones de Java utilizadas son Java 17 y Java 8. Se utilizó el IDE IntelliJ.

2.2.2 Planificación

El trabajo y las implementaciones se llevaron a cabo durante tres semanas:

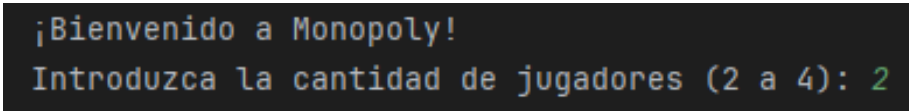
1. La primera semana nos encargamos de empezar con el modelado de las clases Tablero, Jugador, Casilla, Pieza, Propiedad, Carcel, Estacionamiento, Impuesto, ArcaOCasualidad y Adelante.
2. La segunda semana agregamos la interfaz IAccionDinero, decidimos que la clase Casilla sea abstracta y agregamos polimorfismo. Además de las subclases de Propiedad y de Carta(clase abstracta) y la clase Mazo. Por último las relaciones entre las clases.
3. En la tercera semana nos encargamos de realizar la interfaz, terminar el diagrama de clases, hacer el informe, el JavaDoc y la presentación.

2.2.3 Distribución de tareas entre los diferentes integrantes

Durante el transcurso del proyecto el modelado de clases y la implementación de código fue resuelta por ambos integrantes. Luego el archivo.jar fue realizado por Forni Diego y el JavaDoc por Massacesi Juan Ignacio.

2.2.4 Descripción del juego implementado una vez ejecutado

Simularemos una partida con 2 jugadores e iremos comentando lo que vemos a medida que avanza el juego.



```
¡Bienvenido a Monopoly!  
Introduzca la cantidad de jugadores (2 a 4): 2
```

Figura 7. Comienzo del juego

Lo primero que vemos es un cartel de bienvenida e introducción de jugadores en el juego. Se puede jugar de 2 a 4 jugadores.

```
Jugador 1, elige tu nombre:
Juan
Juan, elige tu pieza:
Piezas disponibles:
1. SOMBRERO
2. AUTO
3. PERRO
4. DEDAL
5. BOTA
6. BARCO
7. PLANCHA
8. CARRETILLA
Introduce el número de la pieza que deseas elegir: 1
Jugador 1, ingresa tu posición (entre 1 y 20): 1
```

Figura 8. Interacción con el jugador

Luego pedimos que el jugador ingrese su nombre, elija un número entre 1 y 8 para seleccionar la pieza y la posición (entre 1 y 20) donde comenzará en el tablero. Se muestra lo mismo para el otro jugador.

```
Desea utilizar el juego automático? (Si/No)
No
Modo manual activado.
```

Figura 3. Juego automático

Luego de ingresar los jugadores, piezas y posiciones le preguntamos al usuario si quiere que el juego sea automático o no. En este ejemplo no lo es.

```
-----
Turno de SOMBRERO
Dinero actual: $1000
Presiona Enter para tirar los dados.
```

Figura 9. Información del jugador

Ahora vemos la pieza del jugador que está jugando y su dinero actual. Además de pedirle que presione Enter para tirar los dados.

```
Dado: 4
Avanzas 4 casillas
Casilla actual: Impuestos
```

Figura 10. Información del jugador

Vemos el dado, la cantidad de casillas que avanza y la casilla donde cae el jugador. En el transcurso del juego a la información de la Figura 5 se le agrega un mensaje que indica si el jugador tiene una carta para salir de la cárcel y otro mensaje que nos dirá las propiedades que el jugador posee .

Salida	Av. Mediterráneo	Arca Comunal	Av. Báltica	Impuestos	Estacionamiento
PERRO				SOMBRERO	
Av. Nueva York					Av. Oriental
Av. Tennessee					Casualidad
Arca Comunal					Av. Vermont
Plaza Jaime					Av. Connecticut
El Muelle	Av. Virginia	Av. los Estados	Electricidad	Plaza San Carlos	Ir a la cárcel

Impuesto de \$200 pagado!
Dinero restante: \$800

Figura 11. Tablero y acción realizada por el jugador

Luego de saber donde cayó el jugador se muestra el tablero y la acción que el jugador realiza dependiendo en qué casilla está, hay casillas donde el jugador hará diferentes acciones. Por ejemplo:

```
¿Desea comprar Av. Báltica por $90? (Si/No)
sí
Compra de Av. Báltica realizada.
Dinero restante: $910
```

Figura 12. Acción realizada por el jugador

Aquí se le pregunta al jugador si desea comprar la propiedad o no. Según la respuesta se devuelve un mensaje y se muestra el dinero restante si realizó la compra.

```
Renta de $45 pagada con éxito.
Dinero restante: $505
Diego Ha recibido: $45
```

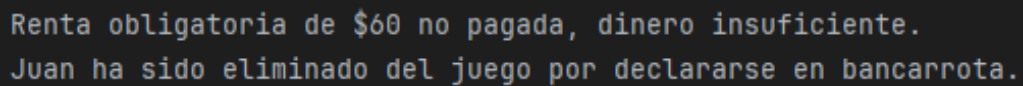
Figura 13. Acción realizada por el jugador.

Aquí el jugador paga una renta de manera automática, se muestra su dinero restante, además de ver quien recibe el dinero de la renta (el propietario).

```
¡Ve a la cárcel!
Multa cobrada, sale de la cárcel
Dinero restante: $235
```

Figura 14. Acción realizada por el jugador

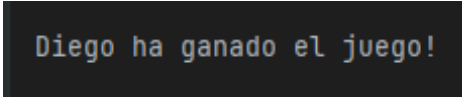
Aquí el jugador está en la cárcel y paga automáticamente la multa y se muestra su dinero restante.



```
Renta obligatoria de $60 no pagada, dinero insuficiente.  
Juan ha sido eliminado del juego por declararse en bancarrota.
```

Figura 15. Jugador eliminado del juego

Si se trata de un pago obligatorio ya sea una multa, renta, impuesto o que una carta lo indique y el jugador no tenga el dinero suficiente para realizarlo, automáticamente el jugador pasa a quedar en bancarrota, se lo desposee de sus propiedades y es eliminado del juego.



```
Diego ha ganado el juego!
```

Figura 16. Jugador ganador del juego

Por último, en el caso de que quede un solo jugador sin estar en bancarrota, el juego termina y ese jugador es el ganador.

3. Resultados

En este trabajo, se lograron cumplir todos los requisitos establecidos e incluso se superaron en varios aspectos. Se implementan las funcionalidades esenciales del juego de Monopoly, así como varias características adicionales que enriquecen la experiencia del usuario.

- **Cumplimiento de Requisitos:** Se logró inicializar el tablero con el número adecuado de jugadores, y se implementó un sistema para listar sus nombres, tipos de piezas y posiciones actuales en el tablero. La simulación de al menos 10 vueltas completas se realizó exitosamente, y el sistema informa de manera detallada lo que ocurre en cada tirada de dados.
- **Visualización del Tablero:** Se desarrolló una forma sencilla pero efectiva para visualizar el estado del tablero, mediante una representación visual por consola, lo que facilita la comprensión del juego y la gestión de las acciones de los jugadores.
- **Gestión de Bancarrota:** Se incluyó una lógica para determinar si un jugador está en posibilidad de bancarrota, proporcionando información crucial para la gestión del juego y asegurando que los jugadores puedan recibir alertas sobre su estado financiero.
- **Interfaz y Polimorfismo:** Se implementó la interfaz `IAccionDinero` en el diagrama de clases, y se incorporaron múltiples métodos polimórficos, lo que permitió una mayor flexibilidad y extensibilidad en el diseño del sistema.
- **Características Adicionales:** Además de los requisitos principales, se desarrollaron componentes adicionales que enriquecen la experiencia de juego. Esto incluye la funcionalidad de juego automático y otros elementos propios de Monopoly que no estaban especificados en los requisitos iniciales pero que aportan valor al juego.

En resumen, se logró crear una experiencia de juego simple y disfrutable, superando las expectativas iniciales y proporcionando un sistema robusto y funcional que captura la esencia del Monopoly mientras ofrece mejoras y características adicionales.

Salida	Av. Mediterráneo	Arca Comunal	Av. Báltica	Impuestos	Estacionamiento	
	PERRO					
Av. Nueva York					Av. Oriental	
Av. Tennessee					Casualidad	
Arca Comunal					Av. Vermont	
CARRETILLA						
Plaza Jaime					Av. Connecticut	
El Muelle	Av. Virginia	Av. los Estados	Electricidad	Plaza San Carlos	Ir a la cárcel	
AUTO			DEDAL			

Figura 17. Tablero de juego con 4 jugadores

4. Ejecución

En la carpeta entregada, se encuentra el archivo monopoly.jar, este es el archivo ejecutable. Para poder jugar, tiene que abrir la consola y usar el comando: `java -jar <dirección del archivo>`, para encontrar esta dirección, varía según el sistema operativo

Windows:

Navega hasta la carpeta donde está guardado el archivo monopoly.jar usando el Explorador de Archivos.

Haz clic en la barra de direcciones en la parte superior (donde muestra la ubicación actual de la carpeta).

Copia la ruta completa que aparece.

En la consola, cuando vayas a usar el comando `java -jar`, agrega la ruta copiada, entre comillas si contiene espacios.

Comando:

```
java -jar "C:\ruta\hacia\monopoly.jar"
```

Linux:

Abre un explorador de archivos o navega hasta la carpeta usando la terminal.

Para obtener la ruta completa de un archivo, puedes hacer clic derecho sobre el archivo monopoly.jar y seleccionar la opción "Copiar ruta" o similar, dependiendo del administrador de archivos que uses. Otra opción, es abriendo propiedades.

También puedes usar la terminal, navegando hasta la carpeta con el comando `cd` y ejecutando `pwd` para mostrar la ruta completa:

```
cd /ruta/a/la/carpeta
```

```
pwd
```

Luego, usa esa ruta en el comando `java -jar` en la terminal.

Comando:

```
java -jar /ruta/hacia/monopoly.jar
```

Mac:

Abre Finder y navegue hasta la carpeta donde se encuentra el archivo `monopoly.jar`.

Haz clic derecho en el archivo y selecciona la opción "Obtener información" (Get Info).

En la ventana de información, verás un campo llamado "Dónde" (Where), que muestra la ruta del archivo. También puedes mantener presionada la tecla "Opción" (Option) y hacer clic derecho en el archivo para que aparezca la opción "Copiar como ruta".

Usa esa ruta en la terminal con el comando `java -jar`.

Comando:

```
java -jar /Users/usuario/monopoly.jar
```

5. Conclusiones

A pesar de la aparente sencillez del juego de mesa Monopoly, este trabajo reveló su verdadera complejidad al explorar en profundidad sus mecanismos y estructuras. Se desarrollaron 20 clases únicas, cada una con relaciones intrincadas que reflejan la riqueza y la dinámica del juego. La implementación de estas clases y sus interacciones demostró ser un desafío significativo, pero a la vez resultó en una solución efectiva y funcional.

El trabajo logró superar varios desafíos extensos, destacando especialmente en el aspecto visual del juego ejecutado por consola. La creación de una representación clara y comprensible del tablero, junto con la implementación de características adicionales, contribuyó a una experiencia de usuario satisfactoria y enriquecedora.

Además, a lo largo del desarrollo, se pusieron en práctica todos los conocimientos adquiridos en las unidades 1 y 2 de la materia "Paradigmas de la Programación". La aplicación de conceptos fundamentales de programación orientada a objetos, como la encapsulación, herencia, polimorfismo e interfaces, permitió crear un sistema robusto y flexible. Esta experiencia consolidó los principios aprendidos en el curso, demostrando su aplicabilidad en el desarrollo de proyectos reales.