UNIVERSIDAD DE SANTIAGO DE CHILE
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA INFORMÁTICA

Laboratorio Nº 2 : Implementación de “Capitalia” bajo el paradigma lógico.

Nombre: Felipe Aedo Jaramillo
Profesor: Edmundo Leiva
Asignatura: Paradigmas de Programación
6 de junio de 2025

Índice

Introducción

El presente informe tiene como objetivo presentar y analizar la implementación del juego de mesa multijugador ***Capitalia***, una versión inspirada en el clásico *Monopoly*. Este desarrollo surge como una propuesta académica para explorar la representación del conocimiento, la toma de decisiones y la evolución de estados dentro de un sistema de reglas bien definidas, todo ello enmarcado en un contexto lúdico y reconocible.

La implementación de ***Capitalia*** se realizó siguiendo el paradigma de programación lógica, el cual se centra en la definición de relaciones y hechos mediante lógica formal, en lugar de instrucciones secuenciales. Este enfoque permite modelar de forma declarativa los distintos elementos del juego, como propiedades, jugadores y acciones, facilitando una estructura clara y flexible. Para ello, se utilizó el lenguaje Prolog, ampliamente conocido por su capacidad de representar conocimiento y resolver problemas mediante inferencia lógica y backtracking.

La estructura del informe se conforma de la siguiente manera: en primer lugar, una breve descripción del problema, seguida por la descripción del paradigma utilizado. A continuación, se procede con el análisis del problema para dar paso al diseño de la solución con sus aspectos varios. Finalmente, se adjunta el manual de uso para el usuario y los resultados obtenidos, para finalmente dar con las conclusiones.

Descripción del problema

El juego ***Capitalia*** consiste en una simulación económica por turnos en la cual los jugadores compiten por adquirir propiedades, construir en ellas y cobrar rentas a otros jugadores que caigan en sus casillas. El objetivo principal es llevar a la bancarrota a los oponentes mediante una gestión estratégica de recursos y decisiones acertadas.

El objetivo principal del proyecto es desarrollar una versión completa y funcional del juego bajo el paradigma de programación lógica, representando el estado del juego (tablero, jugadores, propiedades, dinero, etc.) de forma práctica, además de incluir la lógica de las reglas de ***Capitalia***. Las principales características a implementar incluyen:

- Transacción de propiedades: Compra e hipoteca.
- Tablero y lógica de movimiento: Movimiento circular a través de un tablero compuesto por cinco tipo de casillas diferentes (salida, carcel, suerte, comunidad, propiedad).
- Implementación de acciones como construcción de casas y hoteles.

- Rentas e impuestos: Por caer en una propiedad ajena, y por completar una vuelta en el tablero respectivamente.
- Implementación de al menos 20 eventos especiales mediante cartas: “Ir a la cárcel”, “Jugada perfecta”, “Ganar lotería”, “Terremoto”, “Bono equitativo”, “Liberar prisioneros”, etc.
- Detección de bancarrota y condiciones de finalización del juego.

Descripción del paradigma

El paradigma de programación lógica es un modelo que concibe el software como un conjunto de hechos y reglas lógicas, a partir de los cuales se realizan inferencias para resolver problemas. En este enfoque, los programas no describen *cómo* realizar una tarea paso a paso, sino *qué* condiciones deben cumplirse para que algo sea verdadero.

Este paradigma se basa en principios clave como:

- **Hechos y reglas**: los programas se estructuran como una base de conocimiento con hechos que representan verdades simples y reglas que describen relaciones lógicas más complejas.
- **Declaratividad**: el programador especifica las relaciones lógicas entre entidades en lugar de algoritmos de ejecución. La resolución queda a cargo del motor de inferencia.
- **Inferencia automática**: los programas se ejecutan mediante mecanismos de deducción (como la unificación y la búsqueda), que encuentran soluciones a partir de las reglas declaradas.
- **No determinismo**: el motor lógico puede explorar múltiples caminos de ejecución, buscando todas las soluciones posibles a una consulta dada.
- **Backtracking**: cuando una vía de solución no conduce a un resultado válido, el sistema retrocede automáticamente para explorar alternativas.

Este paradigma permite que la implementación de ***Capitalia*** represente el conocimiento del juego como una base lógica de relaciones entre jugadores, propiedades y acciones. Las decisiones del juego se modelan como consultas que derivan nuevas verdades a partir del estado anterior, facilitando así un diseño robusto, flexible y basado en la lógica formal.

Análisis del problema

El desarrollo de ***Capitalia*** en Prolog presenta diversos desafíos técnicos y de diseño, particularmente al trabajar dentro de las restricciones del paradigma lógico:

Diseño de las estructuras de datos:
Representar de manera adecuada los datos mediante **TDAs** (Tipos de Datos Abstractos) resulta esencial, pero también desafiante, dado que el juego requiere múltiples estructuras relacionadas (jugadores, propiedades, cartas, tablero, etc.). La lógica del juego debe ser rigurosa y coherente durante la ejecución de los turnos, por lo que un diseño sólido de los TDAs y sus operadores es clave para garantizar la correcta evolución del estado del juego.

Gestión de turnos y flujo del juego:
El control de los turnos debe mantenerse sincronizado con las acciones del usuario, evitando errores como jugadas fuera de turno o saltos indebidos. Asimismo, el sistema debe manejar diversos escenarios de juego, como encarcelamiento, cobro de rentas, cobro de impuestos, cartas de azar, bancarrota del jugador, embargo de propiedades, etc. Definir los casos de fallo es clave para asegurar que cada evento altere el estado del juego según las reglas definidas, incluso arrojando una copia del juego sin alterar cuando esta no puede llevarse a cabo.

Interacción con el usuario:
Al no contar con una interfaz gráfica integrada, la interacción con el usuario en ***Capitalia*** se gestiona a través de consultas y scripts escritos en Prolog. Estos scripts simulan el desarrollo de partidas y permiten verificar el correcto funcionamiento de las reglas del juego bajo distintos escenarios. Dado que el paradigma lógico se basa en relaciones y no en secuencias imperativas, es crucial definir consultas bien estructuradas que representen acciones como lanzar el dado, mover un jugador o aplicar los efectos de una carta. Además, los scripts deben contemplar tanto funcionalidades habituales como situaciones límite, como el encarcelamiento o la bancarrota producto de la incapacidad de pagar una deuda, asegurando así la robustez del sistema.

Limitaciones del entorno:
El desarrollo en Prolog impone ciertas restricciones que requieren una planificación cuidadosa. Al tratarse de un lenguaje lógico, no se trabaja con variables mutables ni estructuras que evolucionan paso a paso, como en paradigmas imperativos o funcionales. En cambio, cada consulta genera un nuevo resultado lógico sin alterar el conocimiento base. Esto implica modelar el estado del juego de forma explícita en cada paso, utilizando hechos y reglas que permiten describir cómo evoluciona el sistema sin modificar directamente sus componentes. De este modo, la lógica del juego debe ser expresada mediante predicados que relacionen estados anteriores con nuevos estados posibles, respetando la consistencia declarativa del paradigma lógico.

Diseño de la solución

El diseño de la solución para ***Capitalia*** se basa en una arquitectura modular declarativa estructurada mediante predicados lógicos organizados en torno a cinco TDA (Tipos de Datos Abstractos) principales, junto a un archivo auxiliar con predicados de apoyo. Cada TDA se define como un conjunto de hechos y reglas que representan entidades y operaciones del juego, respetando los principios del paradigma lógico.

- **TDA jugador**: Representa a cada jugador mediante una estructura lógica con identificador único, nombre, saldo, posición en el tablero, propiedades en posesión, estado de encarcelamiento, cantidad de cartas para salir de la cárcel. Los predicados asociados permiten consultar y transformar estas relaciones, como mover al jugador, modificar su saldo, registrar propiedades adquiridas y detectar bancarrota.
- **TDA propiedad**: Modela cada propiedad del tablero, esta contiene; su identificador, nombre, valor, renta, dueño actual, número de construcciones y estado hipotecario. Se definen reglas para calcular la renta, cambiar el propietario, hipotecar propiedades y realizar construcciones.
- **TDA carta**: Representa una carta de evento (ya sea azar o arca comunal), mediante hechos que contienen un mensaje y una acción lógica asociada. Esta acción se representa como un predicado que opera sobre el estado del juego y devuelve un nuevo estado, permitiendo aplicar transformaciones declarativas sin modificar estructuras de forma destructiva.
- **TDA tablero**: Contiene la información estructural del juego, incluyendo las posiciones de propiedades, casillas especiales y mazos de cartas. Predicados auxiliares permiten obtener la casilla correspondiente a una posición, así como recorrer o consultar el orden de las casillas para avanzar jugadores o disparar efectos en el juego.
- **TDA juego**: Actúa como controlador general del sistema. Modela el estado completo del juego, incluyendo jugadores, tablero, cantidad de construcciones máxima, tasa de impuestos y flujo de turnos. Define predicados que coordinan el avance de la partida, validan acciones, aplican eventos y actualizan los estados resultantes conforme a las reglas lógicas del juego.
- **Operadores auxiliares:** Contiene operadores útiles, como myMember, máximo de una lista, generador determinista de números aleatorios, entre otros.

Simulación de Turno y Prueba del Sistema

Para validar el correcto funcionamiento de la lógica central del juego, se desarrollaron tres scripts de simulación que permiten ejecutar una partida completa.

Cada script define explícitamente todos los elementos necesarios del juego: jugadores, propiedades, cartas y tablero.

El predicado principal utilizado en este contexto es *juegoJugarTurno/6*, parte del TDA juego, que encapsula toda la lógica de un turno. Este predicado tiene la siguiente forma:

juegoJugarTurno(Juego, SeedDados, NSeedDados, Accion, Argumento, JuegoActualizado).

- Juego: Estado actual del juego.
- SeedDados: Lista utilizada para generar los dados del turno actual.
- NSeedDados: Lista de semillas para el siguiente turno.
- Accion: Acción que el jugador desea realizar tras moverse (por ejemplo: comprarPropiedad, construirHotel, construirCasa, hipotecarPropiedad, levantarHipoteca, cartaSalirCarcel, pagarMultaCarcel).
- Argumento: Identificador de la propiedad afectada, cuando aplica (utilizado en acciones como construir o hipotecar).
- JuegoActualizado: Estado resultante tras la ejecución del turno.

El flujo general dentro del predicado incluye:

1. Lanzamiento de dados usando *juegoLanzarDados/4* (determinista).
2. Movimiento del jugador según el valor de los dados.
3. Aplicación de efectos según la casilla de destino: pago de impuestos, cobro de renta o activación de una carta.
4. Ejecución de la acción seleccionada por el jugador, si es válida.
    - Acciones como j*uegoAccionComprarPropiedad/2* no requieren argumento (por eso se debe pasar _).
    - Acciones como jue*goConstruirHotel/3*, *juegoConstruirCasa/3*, *juegoHipotecarPropiedad/3* y *juegoLevantarHipoteca/*3 requieren que el argumento corresponda al ID de una propiedad válida.

El uso de este predicado permite mantener una lógica pura y declarativa, generando un nuevo estado del juego tras cada turno sin modificar el anterior, lo que permite seguir el desarrollo de la partida de manera transparente y controlada.

Aspectos de la implementación

El desarrollo fue realizado en **SWI-Prolog (versión 9.3.14)** utilizando exclusivamente predicados definidos de forma pura y estructuras inmutables, en conformidad con el paradigma lógico.

Para simular la interacción del usuario, se diseñaron scripts que formulan consultas a *juegoJugarTurno/6* con distintos parámetros, permitiendo simular turnos completos de manera controlada. Estos scripts permiten validar tanto el comportamiento básico del juego como escenarios límite, incluyendo bancarrota, cárcel, hipotecas, y eventos de cartas.

Cada acción del juego está modelada como un predicado puro que recibe el estado actual y devuelve uno nuevo. Esto garantiza consistencia y facilita la trazabilidad del flujo de ejecución.

La detección de bancarrota se integra al final del predicado *juegoJugarTurno/6*, y se hace mediante el predicado *juegoAvanzarTurno/2*. Este último, se encarga de actualizar el turno actual de forma circular, asignando -1 cuando el juego acaba (solo queda un jugador que no está en bancarrota). La detección de victoria se hace mediante recursión natural; se recorre la lista de jugadores, retirando al jugador de la lista en la iteración actual si éste se encuentra en bancarrota (sin dinero). De esta manera, el juego acaba cuando en dicha lista solo queda un jugador: el ganador.

Respecto a las propiedades, se optó por definir el precio de una casa como el precio de la renta base en la propiedad (este siempre es más bajo), además permitiendo su construcción aún si no se cae en la posición de la propiedad en posesión. Esto le da relevancia a la mecánica de construcción de casas, haciéndola viable para conseguir la victoria. Por otro lado, hipotecar tiene las siguientes consecuencias: El banco le paga la mitad del valor de la propiedad; para levantar la hipoteca, hace falta pagar el valor de la hipoteca + impuestos. Una propiedad hipotecada no cobra renta.

El jugador puede perder sus propiedades y quedar fuera del juego en caso de quedar sin dinero, es por eso que debe anteponerse a esta situación e hipotecar con anterioridad alguna de sus propiedades. El jugador puede volver al juego en caso de conseguir dinero mediante una carta que saque otro jugador.

La representación del tablero se realiza mediante listas de hechos que asocian propiedades y casillas especiales a posiciones numéricas, lo que permite recorrer y consultar el tablero de forma declarativa.

Las cartas de evento están implementadas como hechos con una acción lógica asociada representada mediante predicados con la siguiente estructura: *juegoAccionCarta/2.* Que todas las acciones de cartas sólo requieran del juego de entrada para generar el juego de salida permite ejecutar efectos dinámicos sobre el estado del juego de manera limpia y práctica. Para seleccionar una carta de suerte o comunidad, se hace uso de la primera semilla en la lista de semillas entrantes para elegir una carta de la lista correspondiente, manteniendo un enfoque determinista.

Como detalle adicional, se agregó la funcionalidad *jugadaPerfecta/4,* que recompensa al jugador por su suerte en caso de sacar todos los dados idénticos; le otorga un pequeño bono de dinero y lo libera de la cárcel.

La modularidad del proyecto permite extender el juego fácilmente: se pueden añadir nuevas acciones, tipos de cartas, reglas especiales o incluso mecanismos como subastas, sin comprometer la estructura lógica del sistema.

Instrucciones de uso

1. Asegurarse de tener instalado **SWI-Prolog** (versión 9.3.14).
2. Descargar el archivo comprimido .zip
3. Localizar la carpeta en el ordenador, descomprimir y acceder.
4. Abrir una ventana “cmd” localizada en la carpeta del código fuente
5. Ingresar “swipl” en la terminal para proseguir con la consulta.
6. Ingresar “consult(“script_1_… .pl”).” con el nombre exacto del script.
7. Observar detenidamente el juego resultante.
8. Es posible crear más juegos de prueba si se sigue la estructura de los scripts entregados (necesario importar el módulo main siempre al comienzo).

Resultados

El desarrollo de **Capitalia** es completamente funcional y cumple con todos los **requisitos principales y complementarios** del proyecto. El sistema de turnos se ejecuta de forma correcta, alternando entre jugadores sin errores y gestionando adecuadamente el flujo completo del juego. Las acciones asociadas a las cartas se aplican correctamente al jugador o jugadores correspondientes, modificando el estado del juego de forma coherente con sus efectos definidos.

Las funcionalidades principales (como el cobro de rentas, la compra de propiedades, y la aplicación de eventos especiales como *Ir a la cárcel* o *Suerte*) funcionan según lo esperado. Estas acciones han sido **validadas mediante múltiples scripts de prueba**, que simulan partidas completas y situaciones límite. Estos scripts permiten verificar la robustez del programa y aseguran un comportamiento consistente bajo diferentes configuraciones de tablero y decisiones del jugador.

Entre los aspectos destacados que fueron completamente implementados se encuentran:

- El **sistema de hipotecas**, que permite hipotecar y levantar propiedades siguiendo las reglas del juego.

- Cálculo y cobro de rentas basado en cantidad de casas o existencia de un hotel en la propiedad.
- Cobro de impuestos basados en la renta total del jugador (suma de todas las rentas de las propiedades que posea).
- La implementación de **10 cartas de cada tipo** (Suerte y Comunidad), cada una con efectos únicos y funcionales.
- La **jugada perfecta**, que se activa automáticamente cuando un jugador lanza dados idénticos, otorgando dinero y el derecho a ser liberado de la cárcel.
- El **embargo automático de propiedades** cuando un jugador queda sin dinero suficiente para pagar deudas (bancarrota).

La visualización del estado del juego se actualiza correctamente tras cada acción, lo que facilita el seguimiento del progreso de la partida y permite comprobar fácilmente la **consistencia del sistema** en cada turno.

Además, la **selección de cartas aleatorias** fue diseñada de forma **determinista**, utilizando semillas de entrada para garantizar replicabilidad en las simulaciones de prueba. Esto asegura que los scripts puedan ejecutarse varias veces con resultados idénticos, favoreciendo la depuración y verificación del sistema.

En resumen, **Capitalia** es un sistema completo, robusto y extensible, capaz de simular de forma fidedigna una partida de Monopoly bajo el paradigma lógico.

Conclusiones

El desarrollo de **Capitalia** demostró la viabilidad de implementar un juego de mesa complejo, como *Monopoly*, utilizando exclusivamente el paradigma lógico en Prolog. A pesar de las restricciones propias del lenguaje —como el uso exclusivo de estructuras inmutables, la ausencia de variables globales y efectos laterales— fue posible modelar adecuadamente tanto el estado del juego como sus transiciones, basándose únicamente en predicados puros.

El uso de **Tipos de Datos Abstractos (TDAs)** permitió estructurar el conocimiento del sistema de forma modular, facilitando la organización del código y la implementación de nuevas funcionalidades. Esta separación en componentes bien definidos (jugadores, propiedades, tablero, cartas y juego) otorga claridad, reutilización de código y extensibilidad.

Asimismo, el enfoque declarativo de Prolog favoreció la transparencia de la lógica del juego y una gestión robusta de los estados posibles, lo cual se tradujo en una implementación clara y consistente del flujo de turnos, las reglas y los eventos aleatorios. La utilización de predicados deterministas con semillas de entrada aportó

replicabilidad y control en las simulaciones, lo que facilitó tanto la validación como el debugging del sistema.

Uno de los principales aprendizajes del proyecto fue la importancia del diseño anticipado de estructuras de datos y de la lógica de transición entre estados, especialmente en un entorno donde no existen variables mutables ni efectos secuenciales. También se destacó la versatilidad del paradigma lógico para representar sistemas complejos mediante relaciones formales entre entidades.

Finalmente, el resultado obtenido (un sistema completo, funcional y robusto) demuestra que es posible modelar y ejecutar simulaciones lúdicas de alta complejidad dentro del marco de la programación lógica. Además de cumplir con los requisitos propuestos, **Capitalia** establece una base sólida para futuras extensiones y aplicaciones, tanto en el ámbito académico como en la simulación de procesos de toma de decisiones.

Referencias

[1] SWI-Prolog Official Documentation – https://www.swi-prolog.org/pldoc/

[2] Bratko, I. (2012). *Prolog Programming for Artificial Intelligence* (4th ed.). Pearson.

[3] Sterling, L., & Shapiro, E. (1994). *The Art of Prolog* (2nd ed.). MIT Press.

[4] Hasbro. *Monopoly: Official Rules*.