Informe Laboratorio 2
Paradigma Lógico

Nombre: Ian Leonardo Rioseco Castro
Rut: 21.249.591-3
Profesor: Edmundo Leiva

Contenido

Introducción

En este informe, desarrollaremos como se construyó el juego conecta 4 utilizando el lenguaje de programación prolog, aplicando el paradigma lógico. Conecta 4 es un juego para dos jugadores que se juega en un tablero vertical de 6 filas y 7 columnas [anexo 1]. Cada jugador tiene un color asociado, “rojo” o “amarillo”, y cada jugador tiene 21 fichas para toda la partida. Los jugadores alternan turnos para dejar caer sus fichas en las columnas del tablero, buscando formar una línea de 4 fichas consecutivas del mismo color en sentido vertical, horizontal o diagonal, Si el tablero se llena o ninguno logra formar una línea de cuatro, el juego termina en empate.

El informe esta estructurado por los siguientes capítulos.

- Introducción: Presenta el contexto y objetivo del proyecto.
- Análisis del problema: Desglosa el problema a resolver, sus características y limitaciones.
- Diseño de la solución: Evalúa el cumplimiento de los requisitos funcionales y las pruebas realizadas.
- Conclusión: Discute los beneficio y limitaciones del enfoque lógico aplicado al desarrollo del juego.

Descripción del problema

El problema principal es desarrollar una versión digital del juego conecta 4 implementada en prolog, que permita simular todas las acciones posibles del juego físico: colocar fichas, verificar victorias o empates y gestionar los turnos de los jugadores.

Conecta 4 es un juego competitivo para dos jugadores que se desarrolla sobre un tablero de 6 filas por 7 columnas. Los jugadores alternan turnos insertando fichas de su color en una de todas las columnas disponibles, Estas fichas “caen” hasta la posición mas baja vacía de la columna seleccionada. El objetivo es ser el primero en formar una línea de 4 fichas consecutivas de su color en cualquier dirección, ya sea vertical, horizontal o diagonal [anexo 2].

Características del problema:

- Dimensiones del problema: El tablero tiene una estructura fija de 6 filas por 7 columnas, con un máximo de 42 fichas jugables.
- Reglas básicas: Los jugadores se alternan los turnos insertando una ficha, el juego termina cuando se cumple con alguna de estas condiciones:
    - Un jugador logra formar una línea en cualquier dirección con 4 fichas consecutivas.
    - El tablero se llena y no existe ganador, por lo tanto, termina en empate.

- Limitaciones físicas del problema: una columna no puede aceptar mas fichas si todas sus filas están llenas.

Limitaciones del problema:

- El tablero esta limitado a un área de juego especifica, lo que permite una cantidad de movimientos predeterminada.
- Cantidad de fichas limitada.
- Cumplir con alguna de las condiciones de victoria para que haya ganador.

Descripción del paradigma

El paradigma lógico se basa en una estructura compuesta por Hechos, Reglas y la ejecución mediante consultas. Su principio fundamental es el uso de inferencia lógica para resolver problemas a través de clausulas y relaciones definidas declarativamente. Este enfoque pone énfasis en el "que" se desea lograr, en lugar de enfocarse en el "como" debe lograrse, delegando la resolución al motor lógico del programa.

Un elemento clave en el paradigma lógico es la recursión, que se emplea como la herramienta central para la manipulación de datos especialmente en el manejo de estructuras como lista. Este paradigma se adapta de manera eficiente a la construcción del juego Conecta 4 gracias a sus características declarativas y su enfoque basado en relaciones y reglas.

Análisis del problema

EL problema para resolver en este proyecto es la implementación digital del juego conecta 4 haciendo uso del paradigma lógico en el lenguaje de programación prolog, para resolver esto se requerirá manipulación de listas, creación de predicados lógicos, la constante generación de nuevos hechos basados en los anteriores y el uso de recursión. Esto nos ayudara a simular cada una de las operaciones que se pueden realizar en el Conecta 4 físico y a manejar las diferentes situaciones que pueden surgir durante el juego.

La creación de hechos y reglas será clave para mantener un orden y claridad en el proyecto. Cada uno de estos hechos representara una parte del estado del juego, como los jugadores, el tablero y el historial de jugadas. Las reglas lógicas permitirán manipular estos hechos y realizar las interacciones entre los diferentes componentes del juego, para manejar las diferentes situaciones del juego se hará uso de recursión y manipulación de listas.

Cada predicado o regla tendrá un propósito claro:

- Hechos constructore: Representaran elementos como los jugadores, el tablero, las fichas y el juego.

- Reglas selectoras: Serán usadas para consultar el estado del juego y realizar operaciones como insertar fichas o verificar condiciones de victoria.
- Reglas modificadoras: Permitirán actualizar el estado del juego.

Predicados obligatorios:

- Player hecho constructor que representa el perfil de un jugador, consta de, ID del jugador, nombre, color de la ficha, numero de victorias, numero de derrotas, numero de empates y fichas restantes.
- Piece hecho constructor que representa una ficha, solo necesita el color que usaremos para crear la ficha.
- Board hecho constructor que representa el tablero, consta de 6 filas y 7 columnas, donde cada índice dentro de una fila es una columna.
- Game hecho constructor que representa la creación de una partida, consta de, jugador 1, jugador 2, tablero, turno actual y un historial de movimientos.

Diseño de la solución

Para abordar el desarrollo del juego Conecta 4, se proponen cuatro predicados principales que organizan y gestionan los elementos del juego, implementando estructuras claras con relaciones lógicas, reglas selectoras y reglas modificadoras.

- Player/7: Representa el perfil del jugador. Almacena información como ID, nombre del jugador, color de la ficha, victorias, derrotas, empates y fichas restantes. Proporciona relaciones para consultar y actualizar estadísticas y fichas restantes, además de tener hecho constructor y reglas selectoras y modificadoras.
- Piece/1: define la ficha de cada jugador mediante el color, permitiendo una clara diferenciación entre las fichas de ambos jugadores.
- board/1: Representa el tablero como una lista de listas, organizando las filas y columnas necesarias. Incluye reglas para verificar el estado del tablero, gestionar jugadas, colocar fichas en la posición más baja de cada columna y evaluar condiciones de victoria. Estas verificaciones se realizan mediante reglas que identifican cuatro fichas consecutivas del mismo color en cualquier dirección.
- Game/5: Controla el estado general del juego. Almacena los perfiles de ambos jugadores, el tablero actual, el turno en curso y un historial de movimientos de la partida. Ofrece relaciones para verificar empates, actualizar estadísticas, determinar el jugador actual y finalizar el juego. También incluye reglas para manejar turnos, validar jugadas y actualizar el tablero e historial.

Consideraciones de implementación

El código fue implementando en SWI-Prolog 9.2.5-1 for Microsoft Windows (64 bit), parte del código fue redactado con Notepad++ v8.5.8 (64 bit), y ejecutado mediante SWI-Prolog online (SWISH). Cada predicado principal tiene un archivo .pl para el hecho y sus respectivas reglas.

No se usaron bibliotecas externas del lenguaje.

Instrucciones de uso

Para realizar las pruebas o el uso de este informe, asegurarse de tener un entorno de desarrollo para Prolog. Si estás utilizando **SWI-Prolog** o **SWISH** (el entorno en línea), asegúrate de que esté correctamente instalado y configurado, se deben abrir todos los archivos .pl que se encuentran dentro de la carpeta códigos, para ejecutar las pruebas se debe consultar el archivo tdagame.pl, los scripts de prueba se encuentran separados en 3 archivos .txt, se recomienda leer el archivo léeme.txt.

Para realizar una buena prueba de los scripts, primero se deben abría todos los archivos .pl de la carpeta códigos para esto se debe abrir SWI-Prolog o SWISH, luego ir a file seleccionar edit luego ir a la dirección donde se tienen guardados los archivos.pl [Anexo 3], se debe repetir esto para abrir cada archivo, una vez abiertos se debe ir a file luego a consult [Anexo 4] y seleccionar el archivo tdagame.pl, una ves echo esto se pueden realizar las pruebas de los scripts.

Resultados obtenidos y Autoevaluación

En el proyecto se alcanzaron todos los requisitos funcionales pedidos 18/18, ninguno de los requerimientos funcionales pedidos no se alcanzó, de los 18 requisitos alcanzados no todos tienen un 100% de alcance [anexo 5] debido a las pruebas realizadas ya que a veces entregaban resultados erróneos y que no concordaban con las pruebas realizadas, lo que se utilizó para las pruebas fue el script de prueba entregado en el documento de instrucciones del laboratorio N2.

La creación de las relaciones para verificar victorias utilizando recursión fue particularmente desafiante, debido a las diversas condiciones a considerar, lo que impactó su fiabilidad en todos los casos posibles. Adicionalmente, el predicado de movimiento para los turnos presentó problemas debido a la cantidad de condiciones que debe evaluar, la relación de verificación de empate al momento de hacer las pruebas debió mantenerse en un comentario debido a que entrega false o true, lo que al momento de usar el script de prueba provocaba que la este mismo llegara hasta la verificación de empate o en algunas ocasiones provocaba un bucle de verificaciones de empate, por lo que, no arrojaba los resultados esperados.

Conclusión

Desde un punto de vista técnico, el paradigma lógico presenta los siguientes beneficios y limitaciones en el desarrollo de este proyecto:

Beneficios del paradigma lógico:

- Declarativas: Facilita expresar las reglas del juego de manera natural y modular.
- Unificación y búsqueda automática: Simplifica la implementación de condiciones complejas como victorias y empates.

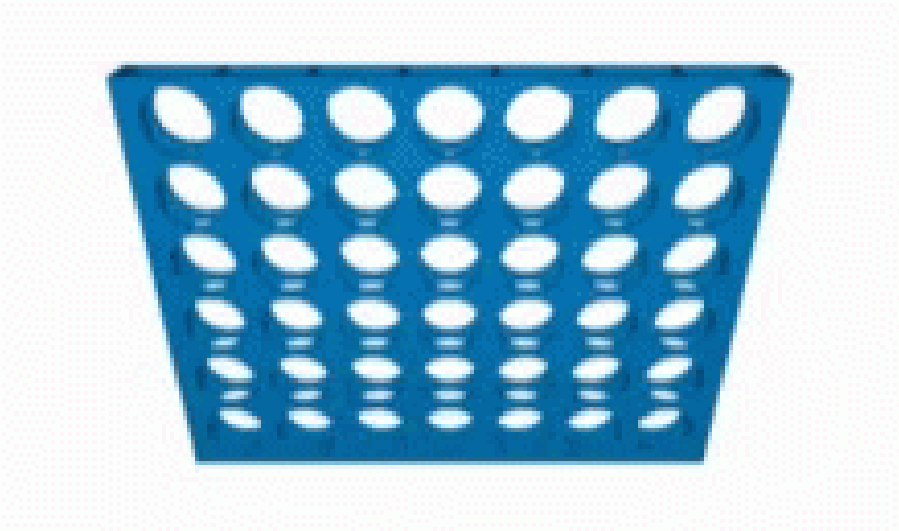
Comparación con otros paradigmas:

En comparación con el paradigma funcional, Prolog permite una mayor claridad en la representación de relaciones y condiciones lógicas. Sin embargo, manejar estructuras complejas como listas puede ser más sencillo en los lenguajes funcionales como scheme, que ofrecen funciones integradas para este propósito. A futuro, una combinación de paradigmas podría mejorar la flexividad y eficiencia del desarrollo.

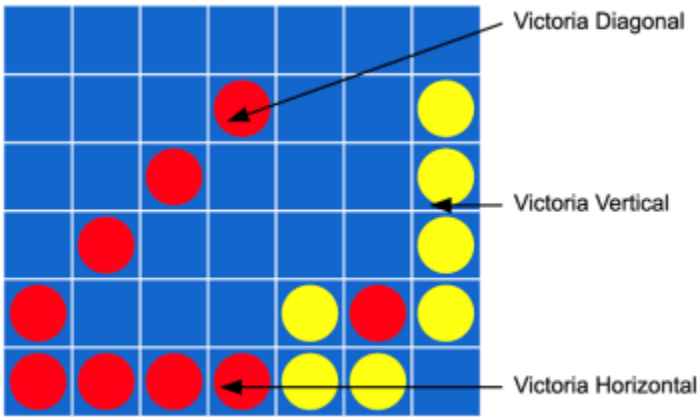
A futuro, una combinación de paradigmas funcionales y lógicos podría ser una opción interesante para mejorar tanto la flexibilidad como la eficiencia del desarrollo. La integración de técnicas de programación funcional podría simplificar el manejo de datos y la manipulación de listas, mientras que el uso de Prolog para la lógica del juego seguiría siendo útil para la verificación de condiciones de victoria y para mantener la claridad en la definición de las reglas.

Anexos

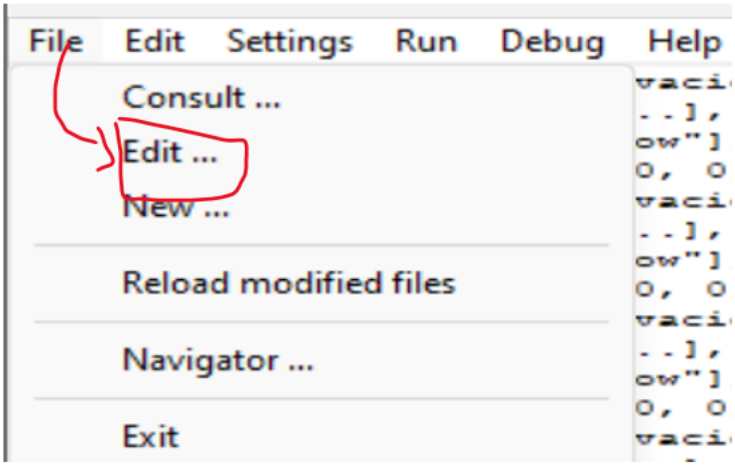
1) Anexo 1



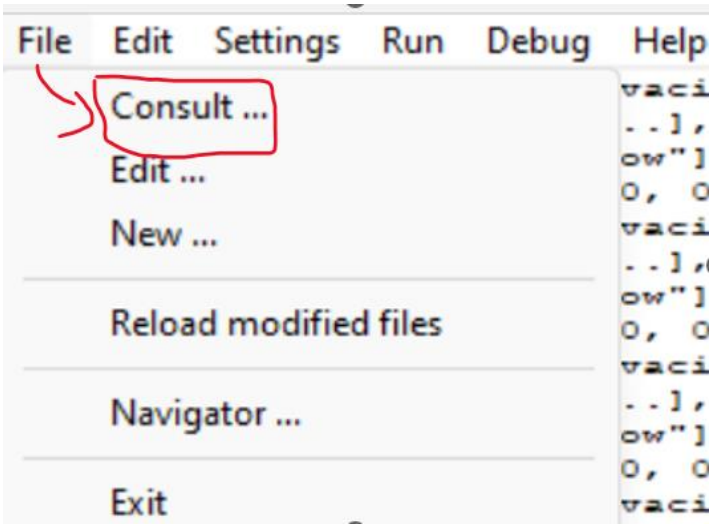
2) Anexo 2



3) Anexo 3



4) Anexo 4



5) Anexo 5

```
Escala:
0: No realizado.
0.25: Implementación con problemas mayores (funciona 25% de las veces o no funciona)
0.5: Implementación con funcionamiento irregular (funciona 50% de las veces)
0.75: Implementación con problemas menores (funciona 75% de las veces)
1: Implementación completa sin problemas (funciona 100% de las veces)

TDA / Función / Puntaje
TDA Player-constructor / player / 1
TDA Player-selector / getIdplayer / 1
TDA Player-selector / getNameplayer / 1
TDA Player-selector / getColorplayer / 1
TDA Player-selector / getWinsplayer / 1
TDA Player-selector / getLossesplayer / 1
TDA Player-selector / getDrawsplayer / 1
TDA Player-selector / getRemainingPiecesplayer / 1

TDA Piece-constructor / piece / 1

TDA Board-constructor / board / 1
TDA Board-otros / can_play / 1
TDA Board-modificador / play_piece / 1
TDA Board-otros / check_vertical_win / 0,75
TDA Board-otros / check_horizontal_win / 0,75
TDA Board-otros / check_diagonal_win / 0,75
TDA Board-otros / who_is_winner / 0,75

TDA Game-constructor / game / 1
TDA Game-otros / game_history / 1
TDA Game-otros / is_draw? / 0,50
TDA Game-otros / update-stats / 1
TDA Game-selector / get-current-player / 1
TDA Game-selector / game-get-board / 1
TDA Game-modificador / end_game / 1
TDA Game-modificador / player_play / 0,75
```

Bibliografía

SWI-Prolog. (s/f). Swi-prolog.org. Recuperado el 27 de mayo de 2024, de https://www.swi-prolog.org/

SWISH -- SWI-Prolog for SHaring. (s/f). Swi-prolog.org. Recuperado el 27 de mayo de 2024, de https://swish.swi-prolog.org/

(S/f). Edu.ar. Recuperado el 27 de mayo de 2024, de https://users.exa.unicen.edu.ar/catedras/prog_exp/apuntes/clase3.pdf