

UNIVERSIDAD DE SANTIAGO DE CHILE
FACULTAD DE INGENIERÍA
Departamento de Ingeniería Informática



CONECTA 4

PROYECTO DE LABORATORIO 2 – PARADIGMA LÓGICO

Paradigmas de Programación

Matias Fernando Ramírez Escobar 21.484.373-0

matias.ramirez.e@usach.cl

Sección: 13204-0 A-1

Fecha de entrega: 29/11/24

Profesor: Edmundo Leiva

**Carrera: Ingeniería de ejecución
en computación e informática**

Tabla de contenidos

Introducción.....	2
El problema.....	3
Scheme y su paradigma	3
Análisis del problema.....	4
Diseño de la solución.....	4
Aspectos de implementación	5
Instrucciones de uso.....	6
Resultados y autoevaluación	7
Conclusiones del trabajo.....	8
Referencias	9
Anexos.....	10

Introducción

En el dinámico y en constante evolución campo de la programación, ciertos de lenguajes han logrado consolidarse como pilares fundamentales en la industria. Sin embargo, la verdadera versatilidad de un ingeniero informático se mide por su capacidad para dominar y aplicar diversos paradigmas de programación según las necesidades del contexto. Este proyecto se centra en la exploración del paradigma lógico a través del lenguaje Prolog, mediante el desarrollo de un innovador juego de

estrategia educativo. La elección de Prolog no es casual, ya que su enfoque declarativo y su poderosa capacidad para gestionar reglas y hechos permiten el diseño de un sistema eficiente y robusto. Este juego tiene como propósito fomentar en los niños habilidades cognitivas, sensoriales, sociales y emocionales, a la vez que estimula el razonamiento lógico, la imaginación y la creatividad.

El problema

El desafío principal de este proyecto consiste en desarrollar una versión del clásico juego Conecta 4 utilizando el lenguaje de programación Prolog. Este reto requiere no solo una representación adecuada del tablero, sino también la implementación precisa de las mecánicas del juego, respetando íntegramente las reglas originales, todo dentro del marco del paradigma lógico. Prolog, aunque no es comúnmente asociado con el desarrollo de juegos interactivos, ofrece una oportunidad única para explorar su potencia y capacidad declarativa en este ámbito, destacando su enfoque en reglas, hechos y sus relaciones, aplicando las herramientas y estrategias propias de la programación lógica. Este enfoque no solo busca demostrar el dominio del lenguaje, sino también su aplicabilidad en la creación de un juego fiel al original. Además, se aprovechan las ventajas distintivas de Prolog, como la simplicidad en la representación de reglas y la automatización en la búsqueda de soluciones, destacando su potencial en escenarios donde el razonamiento lógico es esencial.

Prolog y su paradigma

Prolog, desarrollado en la década de 1970 en la Universidad de Marsella, se distingue por ser un lenguaje centrado en la programación lógica. Este paradigma emplea relaciones y reglas como principales herramientas para manipular datos, mientras que las listas representan una estructura de datos fundamental. A diferencia de los lenguajes tradicionales que utilizan bucles y asignaciones de estado, Prolog se basa en un mecanismo de búsqueda y retroceso (backtracking), lo que permite resolver problemas de manera concisa y declarativa.

La sintaxis de Prolog es clara y expresiva, construida en torno a hechos, reglas y consultas. Esto facilita tanto la lectura como la escritura de programas diseñados para representar y procesar conocimiento. Las consultas en Prolog son tratadas como expresiones lógicas, evaluadas automáticamente para identificar soluciones posibles. Este enfoque lo convierte en una herramienta poderosa para aprender los principios fundamentales de la programación lógica.

Estas cualidades hacen de Prolog un lenguaje idóneo para implementar el juego de estrategia Conecta 4. Su capacidad para modelar reglas y relaciones, junto con la gestión dinámica de posibles jugadas y la verificación de condiciones de victoria, permite desarrollar una versión robusta y funcional del juego, aprovechando al máximo las ventajas del paradigma lógico.

Análisis del problema

La implementación del juego Conecta 4 en Prolog requiere un diseño que abstraiga de manera precisa los elementos y dinámicas del juego, alineándose con las características del paradigma lógico. Esto implica definir y gestionar cada aspecto del juego a través de reglas y consultas bien estructuradas, garantizando tanto la funcionalidad como la eficiencia.

A continuación, se detallan los aspectos clave para simular una partida de Conecta 4 en Prolog:

1. Creación y gestión de partidas:
 - Iniciar partida: Definir los jugadores y establecer el estado inicial del tablero.
2. Gestión de Jugadores:
 - Registro de jugadores: Asignar nombres y colores de fichas a los participantes.
 - Actualización de información: Modificar datos de los jugadores mientras se desarrolla la partida.
3. Desarrollo del Juego:
 - Realización de movimientos: Implementar la colocación de fichas en el tablero respetando las reglas del juego.
 - Validación de movimientos: Verificar que los movimientos sean legales, considerando las posiciones válidas.
 - Detección de resultados: Identificar condiciones de victoria, empate o continuidad del juego.
 - Visualización del tablero: Mostrar el estado actual del tablero tras cada movimiento.
4. Análisis del Juego:
 - Historial de movimientos: Registrar todas las jugadas realizadas en la partida, permitiendo un análisis posterior.

Mientras que las tres primeras categorías son esenciales para la simulación básica del juego, la funcionalidad de un historial de movimientos enriquece la experiencia del usuario. Este predicado permite a los jugadores revisar estrategias, analizar jugadas previas y optimizar su desempeño en partidas futuras, superando las capacidades del juego físico.

Este enfoque integral garantiza una representación lógica y completa del juego, alineada con las capacidades de Prolog y diseñada para ofrecer una experiencia interactiva enriquecedora.

Diseño de la solución

Para representar el juego Conecta 4 de manera efectiva, es esencial identificar los principales Tipos Abstractos de Datos (TDA) que serán implementados mediante

diversos predicados. Con este enfoque, el proyecto se divide en cuatro TDA principales (ver anexo 1): el TDA board (tablero), el TDA piece (ficha), el TDA player (jugador) y el TDA game (juego). El TDA game será el encargado de manipular y coordinar los datos de los otros TDA para simular una partida completa.

Un aspecto importante del diseño es que, al crear un nuevo juego, se requiere un tablero vacío y la definición de los jugadores. Durante cada turno, el jugador debe especificar tanto el jugador activo como la columna en la que desea colocar su ficha, la cual siempre caerá en la última casilla disponible, de arriba hacia abajo. El juego verificará internamente que sea el turno correcto del jugador, y actualizará el estado del juego tras cada movimiento. Estas actualizaciones incluyen:

- Colocar la ficha en el tablero.
- Cambiar el turno.
- Disminuir el número de fichas restantes para el jugador que realizó el movimiento.
- Actualizar el historial de movimientos.
- Verificar si hay un ganador o si el juego ha terminado en empate.
- Si el juego ha concluido, actualizar las estadísticas de ambos jugadores según corresponda.

Para lograr una implementación clara y eficiente, se emplearon diversas estrategias, como la creación de reglas auxiliares dentro de las reglas principales, con el fin de mantener un código limpio y fácil de comprender. Además se utilizó la unificación característica de este lenguaje necesaria para satisfacer los requisitos del proyecto. Esto se realizó respetando siempre las bases del paradigma Lógico.

Aspectos de implementación

Para la implementación de este proyecto se utilizó SWI-Prolog, un entorno destacado para trabajar el paradigma lógico debido a su interfaz interactiva y su lenguaje prolog orientado a este paradigma. Las características de este lenguaje permiten realizar un seguimiento detallado del flujo de ejecución de las consultas al centrarse en que hace un predicado y no en el cómo lo hace, lo que resulta esencial en un proyecto que involucra múltiples Tipos Abstractos de Datos (TDA) y reglas asociadas.

El enfoque de implementación se centra en el encapsulamiento de los distintos TDAs dentro del TDA principal game. Este TDA central coordina los datos de los otros TDA (board, piece, player), lo que asegura que el código sea modular y mantenible. Cada TDA (tablero, ficha, jugador) contiene sus propias reglas para manipular sus datos de forma clara e independiente, lo que favorece la reutilización y la legibilidad del código.

Estructura y Encapsulamiento:

- El TDA game es el núcleo del juego y manipula los TDAs internos.
- Los TDA board (tablero), TDA piece (ficha) y TDA player (jugador) tienen reglas específicas que permiten manipular sus respectivos datos.

La modularidad de este enfoque permite que se pueda modificar o extender el sistema sin afectar las funcionalidades de los demás componentes, facilitando la mantenibilidad y la expansibilidad del código.

Herramientas Utilizadas:

- El proyecto fue desarrollado íntegramente en SWI-Prolog, utilizando la versión más reciente disponible en el momento del inicio del proyecto. SWI-Prolog proporciona herramientas poderosas como:
 - Modo interactivo para probar reglas y consultas de forma rápida.
 - Funcionalidad de trazado que permite analizar la ejecución paso a paso de las reglas y verificar los estados intermedios del programa, facilitando el diagnóstico de problemas durante el desarrollo.

La documentación oficial de SWI-Prolog fue la principal fuente de referencia para comprender las características del lenguaje y optimizar el uso de sus funcionalidades.

Uso de Herramientas de IA:

- Al inicio del desarrollo, se utilizaron herramientas de Inteligencia Artificial para interpretar los errores de sintaxis, lograr adaptar la mente a este nuevo paradigma y los mensajes específicos generados por el intérprete de Prolog. Sin embargo, a pesar del uso de estas herramientas para diagnóstico, no se incorporó código generado automáticamente, garantizando que el desarrollo fuera completamente original y personalizado según los requerimientos del proyecto.

Este enfoque de implementación asegura una estructura sólida y eficiente que facilita la creación, el análisis y la extensión del juego Conecta 4 en Prolog, manteniendo siempre la claridad y la legibilidad del código.

Instrucciones de uso

Este proyecto ha sido diseñado pensando en una fácil ejecución, Una vez abierto el programa WSI-Prolog, se debe consultar el archivo correspondiente y escribir main. Para que se ejecute correctamente la partida de conecta 4 pre-escrita y poder observar su funcionamiento.

1. Instalar WSI-Prolog en el equipo.
2. Abrir la carpeta "**Lab2_214843730_MatiasRamirezEscobar**" y verificar la existencia de los 7 archivos .pl necesarios (ver Anexo 2).
3. Abrir el archivo "**script_base_21484373_MatiasRamirezEscobar.rkt**" en WSI-Prolog.

4. Ejecutar con la consulta (main.).

Los ejemplos incluidos en los scripts de prueba permiten simular diferentes tipos de victorias, como la victoria diagonal, horizontal y un caso de empate. Además, es posible iniciar una nueva partida desde cualquiera de estos scripts. Sin embargo, es importante seguir la estructura de ejecución correcta (ver Anexo 3) para evitar errores.

Se espera que el programa funcione correctamente en todos los Requisitos Funcionales (RF), siempre y cuando el usuario no introduzca datos inválidos. De lo contrario, podrían generarse errores. Algunos ejemplos de situaciones que pueden causar errores incluyen:

- Ingresar incorrectamente el color de la ficha escogida (solo validos los colores "red" y "yellow").
- Proporcionar los parámetros en un orden incorrecto al crear un jugador o al llamar a los predicados "player_play".
- Asignar el mismo color a ambos jugadores.
- Ingresar una columna no válida (un número que no esté entre 1 y 7).
- Intentar ingresar una ficha en una columna que ya este llena.

Resultados y autoevaluación

Los resultados obtenidos respecto a los Requisitos Funcionales (RF) y Requisitos No Funcionales (RNF) se presentan detalladamente en los Anexos 4 en adelante. No obstante, de manera general, se puede afirmar que el programa funciona correctamente bajo la condición de que se respeten el orden de los parámetros y se ingresen los datos válidos, tal como se mencionó en la sección anterior.

Bajo estas condiciones, el proyecto ha demostrado ser exitoso en la implementación del juego Conecta 4 utilizando el paradigma lógico. El código cumple con los objetivos planteados inicialmente, logrando una simulación completa del juego sin presentar errores significativos. Esto confirma que las técnicas empleadas en el diseño y desarrollo del proyecto, como el manejo de Tipos Abstractos de Datos (TDA) y la recursión, fueron adecuadas para modelar de manera efectiva la lógica del juego en Prolog.

En cuanto a la autoevaluación, se considera que se ha logrado cumplir con todos los Requisitos Funcionales esenciales, garantizando un desarrollo estable y eficiente. Los elementos implementados cumplen con las expectativas iniciales del proyecto, ofreciendo una experiencia de juego fluida y sin errores, siempre y cuando los usuarios sigan correctamente las instrucciones de uso. La simulación del juego es precisa, y todas las interacciones entre los jugadores se manejan de manera lógica y coherente, respetando las reglas del juego.

Sin embargo, se identificaron algunos aspectos a mejorar. El manejo de errores en entradas incorrectas podría ser más robusto, ya que, aunque el juego funciona correctamente bajo condiciones ideales, se podría mejorar la retroalimentación en caso de fallos. Además, aunque el código es funcional y claro, se podría optimizar en ciertos aspectos para mejorar la eficiencia. Por último, el juego actual se basa completamente en un sistema de consultas de texto, lo que limita la interacción visual. Para futuras versiones, podría implementarse una interfaz gráfica o un sistema de representación visual del tablero para mejorar la experiencia del usuario.

El proyecto ha sido un éxito en cuanto a su implementación técnica y funcionalidad. Las decisiones de diseño, especialmente el uso de Prolog y la estructuración modular con TDAs, han permitido lograr una solución eficiente y efectiva para modelar el juego Conecta 4 dentro del paradigma lógico.

Conclusiones del trabajo

En conclusión, se puede afirmar que el proyecto ha sido un éxito en su implementación, a pesar de las diferencias entre los paradigmas utilizados por Prolog y Scheme. Prolog, con su enfoque lógico y declarativo, permitió modelar el juego Conecta 4 de una manera eficiente y clara. A diferencia de Scheme, que utiliza un enfoque funcional y declarativo basado en funciones, Prolog se basa en hechos y reglas para definir lo que es verdadero, lo cual representa un cambio significativo en la forma de conceptualizar la lógica del programa.

Una de las principales diferencias entre ambos lenguajes es cómo se gestionan los datos y el control del flujo. En Scheme, se emplean funciones y estructuras como listas para representar el estado del juego, mientras que en Prolog, el estado se representa mediante hechos y variables libres, con un proceso de backtracking que determina cómo se llega a una solución. La comprensión de estos mecanismos de unificación y backtracking fue un desafío, pero una vez asimilados, permitieron desarrollar un programa robusto y bien estructurado.

Finalmente, aunque se siente satisfacción por los resultados obtenidos, se reconoce la importancia de continuar aprendiendo y profundizando en áreas donde Prolog es la base de las nuevas tecnologías como la inteligencia artificial, que se está convirtiendo en una herramienta indispensable en el desarrollo de software. Este proyecto ha servido como un excelente punto intermedio para seguir explorando nuevas tecnologías y paradigmas de programación, abriendo la puerta a futuros desarrollos más complejos y avanzados.

Referencias

Documento-laboratorio-2. (n.d.). *Documentación compartida: Paradigma Lógico*. Retrieved November 28, 2024, from <https://docs.google.com/document/d/1keFuJh6FUPeQ4w0PEXJfTLmy-nY7yTvoNCkBXti8Yfw/edit?tab=t.0>

GitHub · Build and ship software on a single, collaborative platform. (2024). *GitHub*. <https://github.com/>

Prolog Documentation. (n.d.). *SWI-Prolog Documentation*. Retrieved November 28, 2024, from <https://www.swi-prolog.org/pldoc/index.html>

Universitat de Barcelona (UB). (2018). *Memoria del proyecto: Estudio sobre la implementación de sistemas expertos*. Retrieved from <https://diposit.ub.edu/dspace/bitstream/2445/125157/2/memoria.pdf>

Universidad de Santiago de Chile (USACH). (n.d.). *Curso virtual: Introducción a la programación, Paradigma Lógico*. Retrieved November 28, 2024, from <https://uvirtual.usach.cl/moodle/course/view.php?id=10036§ion=16>














Anexos

1.-

Nombre TDA	Dato Abstracto	Estructura	Funciones asociadas al TDA
Board	Representación de un tablero donde se colocarán las fichas y se usara para verificar ganadores u empate.	Lista de Listas de ceros.	board, can_play, play_piece, check_vertical_win, check_horizontal_win, check_diagonal_win, who_is_winner.
Player	Representación de un jugador con sus propios atributos, además de poder actualizar estadísticas en caso de victoria derrota o empate.	Lista que contiene el id, nombre, color, victorias, derrotas, empates.	player, update_stats.
Piece	Representación de una pieza para jugar sobre el tablero que va asociada a cada jugador según su color.	Carácter que corresponde a la primera letra del nombre del color de la ficha del jugador, ej: red -> r	piece
Game	Representación del juego conecta 4, la cual se forma a partir del conjunto de este TDA y los anteriores descritos, el cual puede colocar una ficha y hace las respectivas verificaciones necesarias para el correcto funcionamiento del programa.	Lista contenedora de dos jugadores y un tablero en el cual se juega la partida.	game, is_draw, get_current_player, game_get_board, end_game, player_play, game_history

Tabla de TDAs implementados.

2.-

 .git	25-11-2024 22:28	Carpeta de archivos	
 Autoevaluacion_21484373_MatiasRamire...	27-11-2024 21:33	Documento de te...	2 KB
 board_21484373_MatiasRamirezEscobar.pl	27-11-2024 18:51	Archivo de origen ...	9 KB
 Comandos_21484373_MatiasRamirezEsco...	27-11-2024 21:50	Documento de te...	2 KB
 game_21484373_MatiasRamirezEscobar.pl	28-11-2024 12:13	Archivo de origen ...	6 KB
 Instrucciones_21484373_MatiasRamirezEs...	27-11-2024 22:10	Documento de te...	4 KB
 piece_21484373_MatiasRamirezEscobar.pl	27-11-2024 19:15	Archivo de origen ...	1 KB
 player_21484373_MatiasRamirezEscobar.pl	28-11-2024 12:12	Archivo de origen ...	3 KB
 README_21484373_MatiasRamirezEscob...	27-11-2024 23:06	Documento de te...	1 KB
 Repositorio_21484373_MatiasRamirezEsc...	27-11-2024 19:59	Documento de te...	1 KB
 script_base_21484373_MatiasRamirezEsco...	28-11-2024 11:36	Archivo de origen ...	3 KB
 script1_21484373_MatiasRamirezEscobar.pl	27-11-2024 19:45	Archivo de origen ...	3 KB
 script2_21484373_MatiasRamirezEscobar.pl	28-11-2024 11:31	Archivo de origen ...	3 KB

Carpeta que contiene los archivos .pl.

3.-

```
main :-  
    % 1. Crear jugadores (10 fichas cada uno para un juego corto)  
    player(1, "Juan", "red", 0, 0, 0, 10, P1),  
    player(2, "Mauricio", "yellow", 0, 0, 0, 10, P2),  
    % 2. Crear fichas  
    piece("red", RedPiece),  
    piece("yellow", YellowPiece),  
    % 3. Crear tablero inicial vacío  
    board(EmptyBoard),  
    % 4. Crear nuevo juego  
    game(P1, P2, EmptyBoard, 1, G0),  
    % 5. Realizando movimientos para crear una victoria diagonal  
    player_play(G0, P1, 1, G1), % Juan juega en columna 1  
    player_play(G1, P2, 2, G2), % Mauricio juega en columna 2  
    player_play(G2, P1, 2, G3), % Juan juega en columna 2  
    player_play(G3, P2, 3, G4), % Mauricio juega en columna 3  
    player_play(G4, P1, 3, G5), % Juan juega en columna 3  
    player_play(G5, P2, 4, G6), % Mauricio juega en columna 4  
    player_play(G6, P1, 3, G7), % Juan juega en columna 3  
    player_play(G7, P2, 4, G8), % Mauricio juega en columna 4  
    player_play(G8, P1, 4, G9), % Juan juega en columna 4  
    player_play(G9, P2, 1, G10), % Mauricio juega en columna 1  
    player_play(G10, P1, 4, G11), % Juan juega en columna 4 (victoria diagonal)  
  
    % 6. Verificaciones del estado del juego  
    write('Se puede jugar en el tablero vacío? '),  
    can_play(EmptyBoard), % Si se puede seguir jugando, el programa continuará  
    nl,  
    game_get_board(G11, CurrentBoard),  
    write('Se puede jugar después de 11 movimientos? '),  
    can_play(CurrentBoard),  
    nl,
```

Ejemplo de correcta ejecución del programa (script base).

4.-

Requerimientos	Grado de implementación	Pruebas realizadas	Pruebas exitosas %	Pruebas fracasadas %	Anotación
TDA's	1	4 TDA's implementados	100	0	
TDA Player - constructor	1	20 creaciones de player	100	0	
TDA Piece - constructor	1	38 creaciones de pieza	93	7	Los fracasos fueron cuando se intentó usar un color no permitido.
TDA Board - constructor	1	35 creaciones de tablero	100	0	
TDA Board - otros - can_play	1	8 consultas	100	0	
TDA Board - modificador - play_piece	0,75	35 fichas colocadas	85	15	Los intentos fallidos son de momentos en los que se refinaba la lógica del predicado y el uso del backtracking.
TDA Board - otros - check_vertical_win	1	25 consultas	90	10	Los intentos fallidos son debidos a errores en la lógica, sin embargo, son abordados oportunamente.
TDA Board - otros - check_horizontal_win	1	25 consultas	90	10	Los intentos fallidos son debidos a errores en la lógica, sin embargo, son abordados oportunamente.
TDA Board - otros - check_diagonal_win	1	25 consultas	90	10	Los intentos fallidos son debidos a errores en la lógica, sin embargo, son

					abordados oportunamente.
TDA Board - otros - who_is_winner	1	30 consultas	95	5	Nuevamente fallas con la nueva forma de lógica en prolog, se resuelve oportunamente en conjunto a los anteriores tres RF.
TDA Game - constructor	0,75	20 games contruidos	100	0	
TDA Game - otros - game_history	1	15 llamadas	100	0	
TDA Game - otros - is_draw	0,75	20 verificaciones	100	0	
TDA Player - otros - update_stats	1	15 actualizaciones	100	0	
TDA Game - selector - get_current_player	1	12 llamados a el getter	100	0	
TDA Game - selector - game_get_board	1	13 llamadas al getter	100	0	
TDA Game - modificador - end_game	1	30 términos de un juego	100	0	
TDA Game - modificador - player_play	1	36 veces que se juega	100	0	

Autoevaluación de requerimientos funcionales

5.-

RNF	Implementacion	Anotaciones
Autoevaluación	1	Presente en la carpeta del proyecto.
Lenguaje	1	Lenguaje requerido Scheme/Racket.
Versión	1	6.1 o superior.
Standard	1	Librería estándar.
Documentación	0,75	Toda función presenta su dominio, recorrido, función, tipo de recursión y su tipo, sin embargo, algunas funciones auxiliares no la poseen

Organización	1	Cada TDA tiene su archivo correspondiente.
Historial	1	Se utilizó GitHub respetando los 10 commits mínimos en dos semanas.
Script de pruebas	1	Cada script se ejecuta correctamente.
Prerrequisitos:	1	Cada prerrequisito es implementado por lo que el proyecto está completo.

Autoevaluación de los requerimientos no funcionales