Programación Lógica

Curso 2025

Grupo 01

Informe - Laboratorio 2

Autores:
Gonzalo Apkarian Alonso
5.494.604-2
Diego Pisa Sánchez
5.510.635-2
Leonardo Pesce López
5.471.535-2
Mauricio Morales Gonzalez
5.278.642-4

Profesores:
Luis Chiruzzo
Aiala Rosá
Juan Pablo Conde

FACULTAD DE INGENIERÍA
UDELAR

11 de junio de 2025

1. Módulos

Para este laboratorio no se usaron módulos extra, se implementaron todos los predicados en el mismo archivo principal. Esta decisión se tomó debido a que se tornó una complicación decidir como modularizar de manera eficiente el programa y no se contaba con mucho tiempo, por lo que el equipo se fue por el camino más rápido aunque sin seguir patrones de un buen diseño en su totalidad.

Sin embargo, para facilitar la lectura y corrección del programa, cada vez que se iniciaba el desarrollo de un predicado principal se dejaban líneas con caracteres comentados de forma que se pueda diferenciar fácilmente en el IDE las distintas funciones.

2. Predicados principales

En esta sección se mencionarán y explicarán, además del funcionamiento de los predicados necesarios para que funcione *ceritolog* (es decir, los que vienen explicados en la letra), aquellos predicados auxiliares que el equipo considera esenciales para que el principal pueda realizar su función.

2.1. Tablero

En este predicado simplemente se pide devolver un tablero de tamaño N vacío, donde N es la cantidad de vértices que tiene el tablero por lado, generando una matriz cuadrada donde en realidad quedan $(N-1) \cdot (N-1)$ celdas.

2.1.1. Diseño del tablero

El tablero se crea usando las celdas, la cual es una tupla de 3 elementos (arista superior, arista izquierda, jugador) o (F, C, J). Aquí se tuvo que hacer la primera decisión del diseño del tablero, ya que, si bien el tablero es $N \cdot N$ en vértices, se tomó la decisión de crearlo con $N \cdot N$ celdas para poder tener la información de las aristas de los bordes inferior y derecho. Para esas celdas extra, en la última fila solo se tiene acceso a modificación de la arista superior (que sería la arista inferior del tablero) mientras que en la última columna solo se puede modificar la arista izquierda de esas celdas (que sería la arista derecha del tablero). Lo único negativo de esta implementación es que se guarda una celda en la posición $N \cdot N$ a la que nunca se accede.

A continuación se muestra una foto de como se representa el tablero (extraído de la letra del laboratorio). La fila 4 y columna 4 representan esas celdas 'extra', y la celda $(4, 4)$ a la celda que nunca se accede pero que igual existe por tratarse de una matriz:

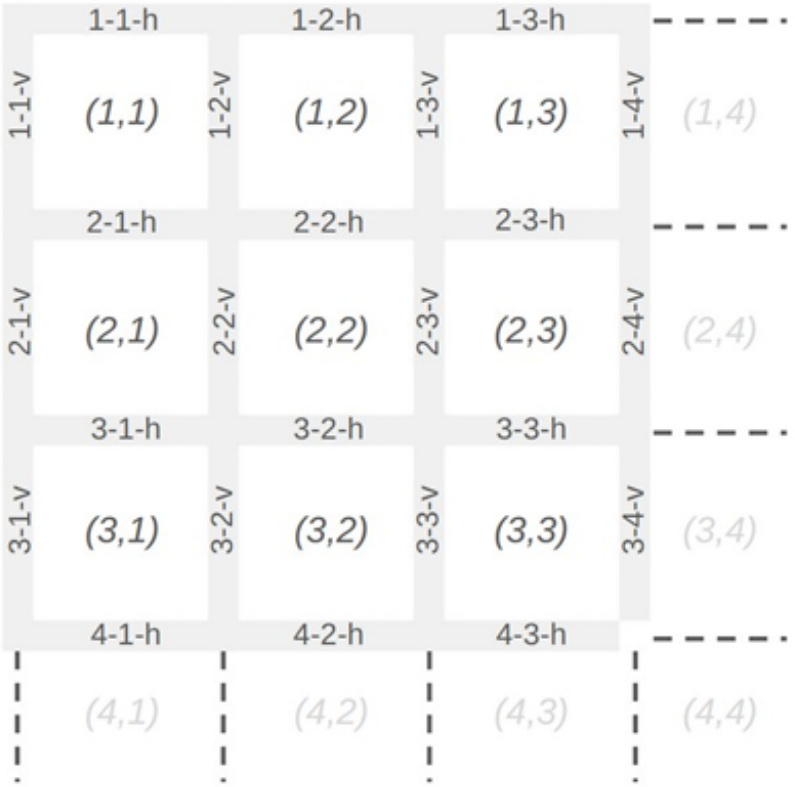

Figura 1: Notar que las líneas punteadas existen en el tablero, pero no pueden ser marcables por los jugadores por estar fuera del rango jugable. Se verifica al realizar la jugada que sea en un lugar válido el movimiento.

2.2. Fin del juego

En este predicado se retorna el final del juego, cuál jugador ganó y con cuantos puntos terminó cada uno. En caso de que no sea el final del juego, el predicado falla.

2.2.1. Diseño de "Fin del juego"

Para el fin del juego se tomó que todas las líneas deben estar marcadas, es decir, una vez que se inicia una partida se debe terminar por completo. Para ello se utiliza un predicado auxiliar que justamente se encarga de verificar que estén todas las líneas marcadas.

2.2.2. Predicado auxiliar 1 - todas_lineas_marcadas

El predicado auxiliar *todas_lineas_marcadas* se encarga de verificar que todas las celdas internas (las que estan entre 1 y $(N-1)$) estén en marcadas como $c(1,1,\_)$, mientras que las de la última fila $c(1,0,0)$ y las de la última columna $c(0,1,0)$. Es importante que las celdas de los bordes derecho e inferior no tengan un jugador que las haya conquistado, de otra manera no estaría funcionando bien el juego.

2.2.3. Predicado auxiliar 2 - contar_puntos

El otro predicado auxiliar que usa esta función es el de *contar_puntos* que pasa celda por celda sumando en el contador del jugador correspondiente si está conquistada por él. Por ejemplo, si se tiene la celda $c(\_, \_, 1)$, se suma un punto al jugador 1. Para que esto funcione se debe verificar antes que el juego está terminado de forma que todas las aristas estén marcadas debido a que $c(1, 0, 1)$ no es una instancia válida de una casilla, así como tampoco lo es $c(0, 1, 1)$ ni $c(0, 0, 1)$ (en todas se usa como ejemplo el jugador 1, pero si en lugar de 1 fuera 2 tampoco serían viables estas instancias).

2.3. Jugada humano

En este predicado se realiza una jugada hecha por un humano al ingresar una fila, columna y dirección que corresponden a la arista de la celda que se desea marcar. La dirección indica cual de las 2 aristas marcar en la celda (h para la superior, v para la izquierda). Finalmente se devuelve el tablero actualizado con el movimiento, de quién es el siguiente turno y una lista de las celdas capturadas en caso de ocurrir una captura (notar que como máximo solo se podrán capturar 2 celdas en una misma jugada).

2.3.1. Diseño de "Jugada humano"

Para la función se realizan las siguientes actividades:

1. Validar la jugada: donde se verifica que la arista seleccionada entra dentro de la zona de juego y que dicha arista no esté ya marcada.

2. Marcar línea: si el movimiento es válido se marca la línea seleccionada.

3. Verificar las capturas: se determinan las celdas que fueron capturadas si correspondiera, puede no haber capturas, una sola o, como máximo, dos celdas capturadas.

4. Determinar siguiente turno: de acuerdo a si se realizó alguna captura o no se determina a qué jugador le toca jugar después. Si se realizó por lo menos una captura, el jugador que acaba de marcar línea sigue jugando.

2.3.2. Predicado auxiliar 1 - jugada_valida

En este predicado se verifica que, en caso de que la dirección sea h, la celda seleccionada caiga entre $(1, 1)$ y $(N, N-1)$ (ya que la línea horizontal de la última columna nunca se marca ya que no hay vértices) y el primer item $c(H, \_, \_)$ sea 0, en otro caso falla porque quiere decir que, o bien la celda no entra dentro del rango mencionado, o bien la línea ya fue marcada. En caso de que la dirección sea v, el rango de celdas va desde $(1, 1)$

y $(N-1, N)$ (última fila no se pueden marcar las líneas verticales) y el segundo item $c(\_, V, \_)$ sea 0.

2.3.3. Predicado auxiliar 2 - marcar_linea

En este predicado simplemente se marca la línea vertical u horizontal según sea el caso y se reemplaza la celda modificada en el tablero nuevo, previamente ya se verificó que la línea seleccionada está disponible.

2.3.4. Predicado auxiliar 3 - verificar_capturas

Para este predicado se deben verificar las celdas afectadas por la línea elegida. Las celdas afectadas pueden ser, si la dirección fue h, la celda donde se modificó (F, C) y la celda de arriba $(F-1, C)$. En este caso se deben verificar un total de 5 celdas, las dos posible afectadas (posible porque puede ser una celda de la primera fila, en ese caso solo hay una celda afectada), una celda a la derecha por cada celda afectada (para verificar si la línea vertical está marcada o no) y, por último, si la celda inferior a la celda que fue modificada tiene la línea horizontal marcada. En el caso de que la dirección fuese v, el pensamiento es análogo, la celda modificada está en (F, C) y la otra posible afectada está en $(F, C-1)$, y se tienen las 2 celdas inferiores a cada una afectada por la línea marcada, además de la que está a la derecha de la modificada.

Para verificar que una celda está capturada se debe dar lo siguiente: para los dos primeros campos de la celda elegida $c(H, V, \_)$ tiene que cumplirse $H = V = 1$, que la celda de la derecha tenga $c(\_, 1, \_)$ y la celda inferior tenga $c(1, \_, \_)$. Si se cumple todo esto, la celda está capturada, por lo que el tercer campo de la celda $c(\_, \_, J)$ J toma el valor del jugador que realizó el movimiento.

2.3.5. Predicado auxiliar 4 - determinar_siguiente_turno

Este predicado simplemente verifica que la lista de celdas capturada esté vacío o no para saber a quien corresponde el siguiente turno, si está vacío al otro jugador, si tiene al menos una captura sigue jugando el mismo jugador que acaba de marcar la línea.

2.4. Jugada máquina

En esta función se debe simular una jugada realizada por una IA con una estrategia *buena y eficiente* la cual culmine antes de los 10 segundos para un tablero con $N = 6$. Esta jugada debe dar la misma información que una jugada humana: un tablero actualizado con el movimiento realizado, de quién es el turno siguiente y una lista con las capturas realizadas por el movimiento en caso de haberlas.

2.4.1. Diseño de "Jugada máquina"

Para implementar está función se hace una atrás de la otra las funciones 'Sugerencia jugada' y 'Jugada humano' de forma que se simula la jugada de un humano a partir del movimiento sugerido.

```
jugada_maquina(Tablero, Turno, Nivel, F, C, D, Tablero2, Turno2, Celdas) :-
    sugerencia_jugada(Tablero, Turno, Nivel, F, C, D),
    jugada_humano(Tablero, Turno, F, C, D, Tablero2, Turno2, Celdas).
```

Figura 2: Notar que los dos predicados utilizados son funciones principales, por lo que tienen una sección particular para cada una ('Sugerencia jugada' se explicará a continuación, mientras que 'Jugada humano' fue explicada en la sección anterior).

2.5. Sugerencia jugada

Para esta función se solicita calcular una buena jugada para sugerirle a un jugador humano. Se sugiere utilizar la estrategia *minimax* utilizada en el curso pero puede ser cualquier estrategia que el equipo elija, el único requisito es que para una tablero con $N = 6$ el tiempo de respuesta sea menor a 10 segundos.

2.5.1. Diseño de "Sugerencia jugada"

Para la implementación del predicado se utilizó la estrategia *minimax* vista a lo largo del curso. Este algoritmo es una regla de decisión que busca minimizar la pérdida en el peor escenario posible mientras maximiza la mínima ganancia en el mejor.

El valor **maximin** se calcula de la siguiente manera:

1. Se chequean todas las posibles acciones de los otros jugadores para cada acción del jugador.

2. Se determina la peor combinación posible de acciones, es decir, aquella que le da al jugador el menor valor.

3. Se determina cuál acción el jugador puede tomar de manera de asegurar que el mínimo valor es el más alto posible.

Definición formal:

$$\underline{v_i} = \max_{a_i} \min_{a_{-i}} v_i(a_i, a_{-i})$$

El valor **minimax** es el menor valor que los otros jugadores pueden forzar al jugador a recibir sin saber las acciones del jugador. Para calcularlo se invierte el orden de la maximización y la minimización, es decir, se minimiza el máximo mientras que en el anterior

se maximizaba el mínimo. La definición formal es:

$$\overline{v_i} = \min_{a_{-i}} \max_{a_i} v_i(a_i, a_{-i})$$

En definitiva, como se trata de un juego de suma 0 (donde gana uno, gana otro o hay empate, es decir $+1$, -1 o 0 de acuerdo a la visión de cada jugador), se diseña una forma de jugar que reduce la ganancia del otro jugador al usar una jugada sugerida, pero también se reduce la ganancia propia por la naturaleza de suma 0 del juego. Dicho esto, se puede argumentar que el diseño de la estrategia es de un jugador defensivo y pasivo, que busca no perder ni arriesgar demasiado.

2.5.2. Predicados auxiliares

Para este algoritmo se utilizan varios predicados auxiliares encargados de hacer lo explicado anteriormente. Se tiene *evaluar_jugadas* que, de acuerdo a los niveles buscados calcula el valor minimax a cada uno para poder determinar cual jugada minimiza la ganancia del otro jugador. Dentro de este predicado se tienen más auxiliares como *valor_minimax* que calcula el valor ya descrito, *mejor_entre* que devuelve cuál valor entre dos jugadas es mejor, y se llama al predicado recursivamente hasta que el nivel sea 0.