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

Paradigmas de Programación

Paradigma imperativo Procedural

FELIPE MATURANA GUERRA

Profesor:
Roberto González

Ayudantes:
Giovanni Benussi
Mauricio Rojas
Esteban Contardo

Santiago - Chile
2-2016

TABLA DE CONTENIDOS

ÍNDICE DE FIGURAS

CAPÍTULO 1. INTRODUCCIÓN

El paradigma “Lógico” el cual es abordado en esta experiencia se basa en los conceptos de lógica matemática, la unidad de programación de éste paradigma se puede reducir a “***Hechos y Reglas”***. Los predicados relacionan a los individuos (no existe el concepto de variables con cambios de estados) permiten realizar ***Deducciones*** de las posibles respuestas a una determinada ***Consulta*** (QUERY).

Al igual que el paradigma Funcional (experiencia pasada) es un paradigma declarativo y trabaja bajo el concepto de unificación.

La característica principal del paradigma consiste en la aplicación de reglas lógicas (*Cláusulas de Horn*) para infrerir conclusiones a partir de datos, el programa asume que toda la información relevante del problema está contenida en los *Hechos*, lo que se conoce como el “***supuesto del mundo cerrado***” lo cual nos obliga a tener cuidado en los hechos y reglas que declaramos ya que de no ser cuidadosos obtendremos deducciones que no necesariamente son correctas al llevarlas a nuestra realidad.

A diferencia de los otros paradigmas vistos, en éste el programador no especifica el cómo se debe resolver el problema, sino que es el sistema quién debe buscar la solución utilizando en primer lugar los hechos y reglas que representan la información del problema.

El problema que se abordará en el siguiente informe consiste en la implementación en lenguaje Swi-Prolog (Paradigma Funcional) el famoso juego “Battleship”, sin embargo, con algunas variaciones respecto a los paradigmas anteriores ya que, si bien el paradigma y el lenguaje permite la creación de tableros, es propicio para realizar consultas como si el disparo en ciertas posiciones es correcto (si le da a un barco, es true), o si un tablero ingresado puede ser considerado como válido o no (si cumple con las condiciones para ser considerado válido).

Se comparará los resultados obtenidos y las pruebas realizadas con las demás experiencias para poder contrastar y conocer las diferencias (ventajas y desventajas) que ofrece cada paradigma.

CAPÍTULO 2. ANÁLISIS DEL PROBLEMA

2.1 BATTLESHIP

Dentro del juego existen requerimientos específicos los cuales fueron considerados en la etapa del diseño de la solución partiendo desde lo más general como es la representación del tablero hasta los resultados de cada disparo efectuado desde el barco, pasando por funcionalidades como la verificación de la creación de tableros y además la visualización de éstos. Para esta experiencia se escogió una representación matricial, la cual se puede implementar en Swi-Prolog mediante listas de listas donde cada lista representa una fila, es decir, una matriz de 6x6 posee la siguiente representación.

```
[[1,2,3,4,5,6],
[7,8,9,10,11,12],
[13,14,15,16,17,18],
[19,20,21,22,23,24],
[25,26,27,28,29,30],
[31,32,33,34,35,36]]
```

Figura 2-1: Representación Tablero.

CAPÍTULO 3. DISEÑO DE LA SOLUCIÓN

El código está estructurado en un único archivo con extensión ".pl" en 2 bloques principales, en primer lugar, se encuentran todos los hechos que permiten un correcto funcionamiento para las consultas a realizar, en especial las que permiten verificar las jugadas a realizar, la creación de los tableros pedidos por la coordinación y los ataques.

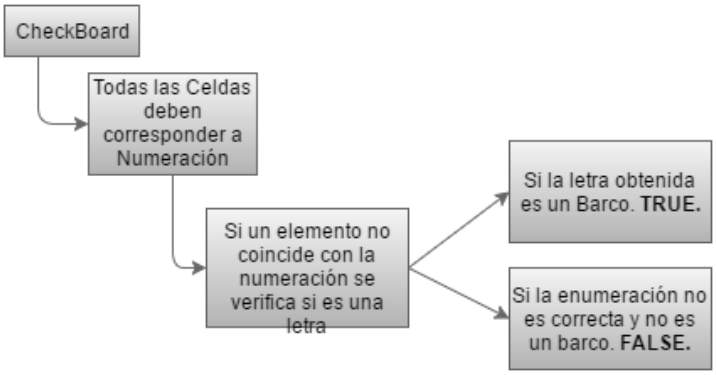

Figura 3-1: Diagrama CheckBoard.

3.1 ENFOQUE DE LA SOLUCIÓN: CREATEBOARD

La representación del tablero será de forma matricial (Listas de listas), la coordinación solicita:

- 3 Tableros de 6x6 con 3 barcos para cada jugador con distintas características
- 2 Tableros de 10x10 con 5 barcos para cada jugador con distintas características
- 1 Tablero de 20x20 con 10 barcos por cada jugador con distintas características

La función **createBoard** recibe las dimensiones de fila y columnas (N y M respectivamente) además de la cantidad de barcos que tendrá cada jugador.

Para la creación de tableros, éstos estarán en la base de conocimiento (HECHOS) de la siguiente forma:

1. Tablero (X, N, M, Nships, BOARD).

Donde X es el número del tablero (0,1,2) cuando las dimensiones son de 6x6, (0,1) cuando las dimensiones son de 10x10 y (0) cuando las dimensiones son de 20x20 es decir, en la base de conocimiento habrá 3 barcos de 6x6, 2 de 10x10 y 1 de 20x20, X es generado de forma aleatoria, de acuerdo a las dimensiones del tablero a crear. El tablero se retorna mediante la variable BOARD.

2. Dimensiones (N, M).

Permite verificar que las dimensiones ingresadas en createBoard correspondan a los tableros (Board) de nuestra base de conocimiento (6, 10 y 20) tanto para las filas como a las columnas.

3. CBarcos (SHIPS, N, M, Y).

Permite verificar que la cantidad de barcos que poseerá el tablero corresponda con las dimensiones del tablero, además de retornar Y, lo cual señala la cantidad de tableros distintos que posee la base de conocimientos que satisfagan a las dimensiones M y N, este parámetro se utiliza luego con la función random para entregar un tablero de forma aleatoria con createBoard.

En el caso de **CheckBoard** recibe sólo el tablero, para comprobar la validez de éste se realizan las siguientes consultas:

1. Se verifica si el tablero está en la base de conocimiento, si es así retorna de forma inmediata TRUE, de lo contrario se realizan las siguientes verificaciones.

2. Se obtiene la cantidad de listas de listas que contiene el tablero, mediante el predicado *largo*, ya que esto representa la cantidad de filas (N) que posee el tablero susodicho.

3. Se obtiene la primera fila de la matriz, para contar la cantidad de columnas (M) que posee el tablero, éstas dimensiones son necesarias para las posteriores verificaciones.

4. Se recorre por completo el tablero, celda por celda, desde la primera (1) hasta la última (N * M). El tablero debe seguir la enumeración correspondiente, aumentando en una unidad mientras aumenta la cantidad de columnas, por lo tanto, cada celda se somete a la revisión si corresponde a la enumeración, o si se encuentra con una letra pregunta si ésta está en la base de conocimientos (letraBarco) la cual va desde la **a** a la **j**.

3.2 ALGORITMOS UTILIZADOS

Para verificar si un elemento pertenece o no a una lista se implementó la regla ***buscar***(X,[X|Xs] , Pos), la cual recorre una lista hasta encontrar el elemento X y retorna la posición en la que éste se encuentre, si no está presente en la lista retorna falso, esta regla utiliza un **algoritmo de búsqueda lineal.**

```
buscar(X,[X|_],0).
buscar(X,[_|T],Pos):- buscar(X,T,Pos2), Pos is Pos2+1.
```

Figura 3-2: Algoritmo de Búsqueda lineal.

CAPÍTULO 4. RESULTADOS

De los requerimientos funcionales obligatorios se cumplió a cabalidad cada uno de los 3 mencionados en el informe. El código es capaz de:

1. Generar tableros válidos de dimensiones, mediante el predicado **createBoard** previamente explicado, 6x6 (3), 10x10 (2) y 20x20 (1) con 3,5 y 10 barcos por cada jugador respectivamente, con distintas características (tamaño, letra que lo representa, etc). Posicionando los barcos tanto del jugador como los del enemigo.

   La operación de generar tableros se probó con cada uno de los tableros que están en la base de conocimientos, al poner intentar crear un tablero que no esté en la base de conocimientos el predicado se vuelve falso ya que, bajo el supuesto del mundo cerrado, toda la información importante está en la base de conocimientos y un tablero de 12x12 no está, por lo cual no lo puede crear.

2. Verificar las características de un tablero, mediante el predicado **checkBoard**, cabe destacar que cualquier tablero puede ser sometido a esta verificación, no sólo los que están en la base de conocimientos, verificando características esenciales para nuestra representación. En el programa se presentan tableros válidos e inválidos para que se pueda probar el predicado.

   La operación de verificar el tablero se probó tanto para tableros válidos y no válidos, que pertenecieran a la base de conocimientos y otros que no. De los tableros en la base de conocimiento la cantidad que fue pedida por la coordinación corresponden a tableros válidos, además se agregaron 2 tableros por cada dimensión los cuales cuentan con características que los hacen inválidos. Existe un caso el cual no fue considerado, no se verifica el correcto posicionamiento de los barcos tanto del jugador como del enemigo, es decir, un barco puede ser posicionado en diagonal e incluso puede un barco estar repartido por partes en el tablero.

3. Play: Permite verificar si los disparos efectuado en las posiciones dada efectivamente dan en un barco, retornando true si todas éstas impactan en un barco enemigo. Además de comprobar que el disparo se realice en la mitad correspondiente al enemigo.

   La operación de verificar si una jugada es válida o no, se probó con distintos tipos de tableros y en posiciones únicas o en varias de ellas de forma simultánea, e incluso ingresando posiciones inválidas.

4. FUNCIONALIDADES EXTRAS.

1. BoardToString: Predicado que permite mostrar por pantalla el tablero según nuestra representación, la funcionalidad fue probada tanto con tableros de la base de conocimiento como tableros ingresados de forma manual (que no perteneces a la base de conocimientos), el algoritmo recorre cada celda y pregunta por la naturaleza del elemento en la base de conocimientos, si es una letra verifica que pertenezca a los barcos en la base de conocimientos (de la ***a*** a la ***j***) si es así, imprime la letra del barco, de lo contrario imprime un espacio vacío separado con '|'.

2. GetScore: Predicado que permite obtener el puntaje de un tablero dado, sin embargo éste debe ser ingresado de forma manual ya que los tableros que están en la base de conocimientos no poseen jugadas en el tablero. La fórmula que calcula el puntaje es la siguiente:

$$score = DE * 100 + BD * 400$$
$$if\ score < 0 \rightarrow score = 0$$

Figura 4-1: Fórmula de cálculo Puntaje.

CAPÍTULO 5. CONCLUSIÓN

El lenguaje de programación, o más bien, el paradigma resultó ser el menos efectivo hasta ahora, para efectos de creación de una estructura como un tablero en su representación matricial ya que éste no es el objetivo principal del paradigma, además que el hecho de no existir variables (que puedan mutar) dificulta la característica de creación y posterior modificación como se utilizó en el primer laboratorio (Paradigma imperativo). Por otro lado, la acción que más se vio favorecida fue la de verificar el estado de un tablero y si éste posee o no las características necesarias para ser considerado válido, el paradigma resultó ser el más propicio para las verificaciones ya que éste es su núcleo de programación, la lógica, determinar si una consulta es verdadera o falsa.

Se cumplieron tanto los objetivos generales, como los particulares de la experiencia, principalmente el complemento que existe de lo visto en cátedra, complementado con la parte práctica vista en este laboratorio mediante la implementación del juego Battleship, o más bien, de realizar las consultas pertinentes de la estructura del juego y las principales reglas para asegurar un correcto funcionamiento de la partida, además de comprobar el gran potencial que nos ofrece esta nueva herramienta adquirida y los principales conceptos de la programación lógica y el lenguaje de programación PROLOG, el cual se complementa muy bien con el conocimiento que se requiere para desarrollar bases de datos en las asignaturas a futuro dentro de la malla de Ingeniería Informática.

CAPÍTULO 6. REFERENCIAS

Hasbro. (2012). *Reglas para 2 Jugadores BATTLESHIP*. (Recuperado 2016). http://www.hasbro.com/common/documents/dad261471c4311ddbd0b0800200c9a66/DC43DE785056900B109B0A81EE470A9E.pdf

Bernal, J (2013). *LENGUAJE IMPERATIVO O PROCEDURAL*. (Recuperado 2016). http://lenguajesdeprogramacionparadigmas.blogspot.cl/2013/03/lenguaje-imperativo-o-procedural.html

AHO, A., HOPCROFT, J.; ULLMAN, J. D. (1987*). Data Structures and Algorithms Addison-Wesley*. Estados Unidos de América.

Kölher, J. (2015). *Apuntes de la Asignatura Análisis de Algoritmos y Estructuras de Datos.* Chile.