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

Paradigmas de Programación

Paradigma Funcional
Manual de Usuario

FELIPE MATURANA GUERRA

Profesor:
Roberto González

Ayudantes:
Kevin Álvarez
Dania Montanares
Pablo Ulloa
René Zarate

Santiago - Chile
2-2016

TABLA DE CONTENIDOS

ÍNDICE DE FIGURAS

CAPÍTULO 1. INTRODUCCIÓN

Battleship es un juego de mesa también conocido como “Batalla de mesa”, es un juego de ingenio y estrategia el cual consiste en un tablero dividido en dos mitades, una de estas mitades corresponde al jugador y la otra al enemigo. Históricamente este juego se jugaba con lápiz y papel tras ser comercializado en 1943 por Milton Bradley Company, sin embargo, no fue hasta 1967 que fue posible conocerlo tal y como lo conocemos, en tres dimensiones. El objetivo del juego es derribar todos los barcos enemigos antes que él nos destruya los nuestros, aquí es donde se pone a prueba nuestro ingenio.

Cada jugador puede ver en su totalidad su tablero, sin embargo, el tablero enemigo sólo se vislumbrarán los lugares donde se han efectuado jugadas (disparos), es decir, antes de comenzar a jugar veremos sólo espacios en blanco en el tablero enemigo.

Antes de comenzar a jugar es necesario posicionar los barcos que tenemos a nuestra disposición en las casillas dentro de nuestro tablero, los barcos pueden estar orientados de forma vertical u horizontal nunca en diagonal.

1.1 REGLAS

Un jugador no podrá realizar dos jugadas en forma consecutiva.

Un jugador no podrá realizar jugadas sobre su mismo tablero ni en una posición dónde ya se ha realizado un disparo.

La dimensión del tablero en general, deberá tener una cantidad par de columnas ya que éste será dividido en dos y a cada jugador le corresponderá una misma cantidad de celdas.

1.2 CÓMO JUGAR

Una vez que se decida quién juega primero, se jugará por turnos, señalando la posición en la cual se realizará un disparo y, en esta implementación, desde qué barco se realizará el disparo (y si es un ataque especial) hasta que alguno de los jugadores no le queden barcos disponibles.

CAPÍTULO 2. PROCESOS DE COMPILACIÓN

2.1 CÓMO COMPILAR Y EJECUTAR

Este proyecto se realizó en el lenguaje de programación **Prolog**, a través de la implementación **SWI-Prolog** de código abierto (Open Source), la implementación del programa está hecha respetando al 100 % el paradigma de programación funcional, del cual se trata esta experiencia. Al ser un lenguaje de programación interpretado, no pasa por un proceso de compilación como el que se explicó en el primer informe (C).

Para poder ejecutar el programa es necesario tener **Swi-Prolog** instalado o **P#**, y tener el archivo "battleShip_189714319_MaturanaGuerra.pl" el cual está adjunto al informe.

2.2 WINDOWS

Las imágenes expuestas a continuación se consiguieron en ambiente Windows, ambiente donde fue creado. Sin embargo, no existen mayores problemas en términos de compatibilidad de Linux ya que es un archivo de texto plano que no utiliza librerías especiales.

Buscamos **SWI-Prolog** en nuestro ordenador y lo ejecutamos.

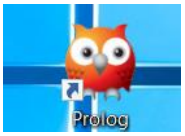

Figura 2-1: Prolog en el escritorio

Una vez tengamos abierto **Prolog**, buscamos en la esquina superior izquierda FILE y luego Consult.

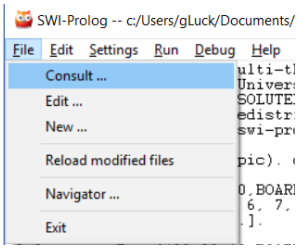

Figura 2-2: Consultar en Prolog.

Una vez abierto la ventana debemos buscar nuestro archivo con extensión “.pl”.

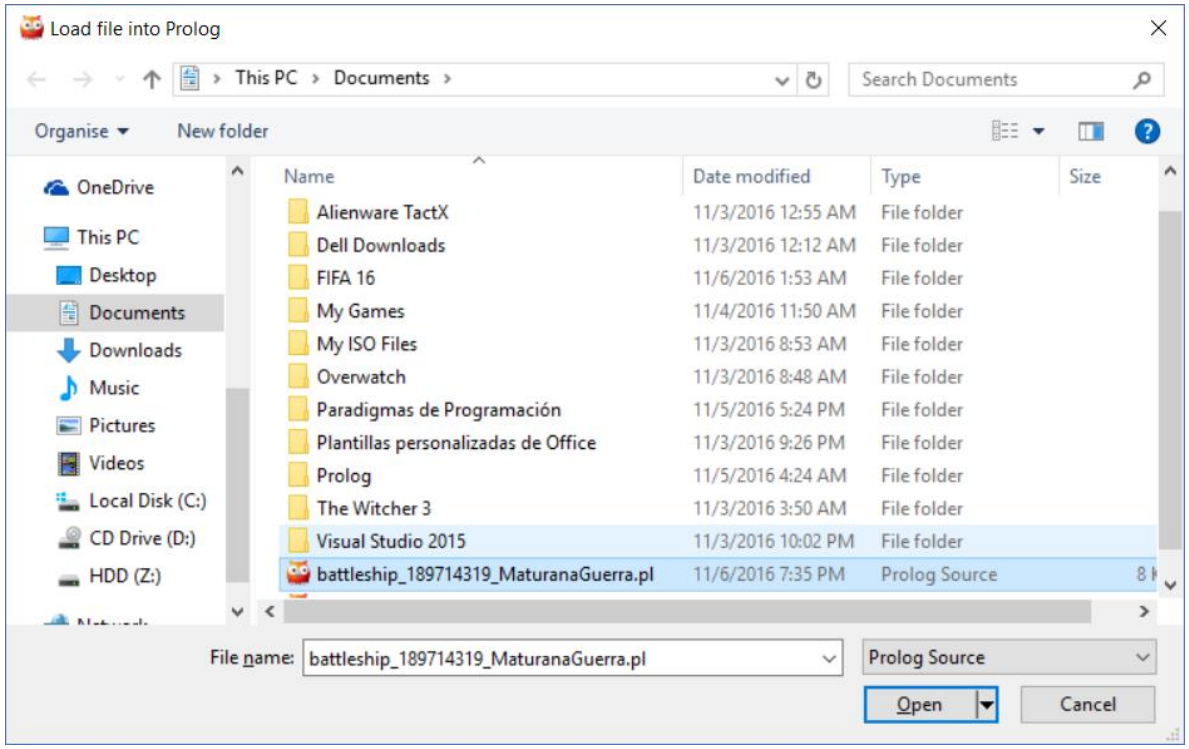

Figura 2-3: Abrir Archivo Prolog.

Una vez abierto, se nos abrirá una ventana donde podremos realizar nuestras consultas, como por ejemplo la de crear un tablero, el tablero es devuelvo en la variable BOARD.

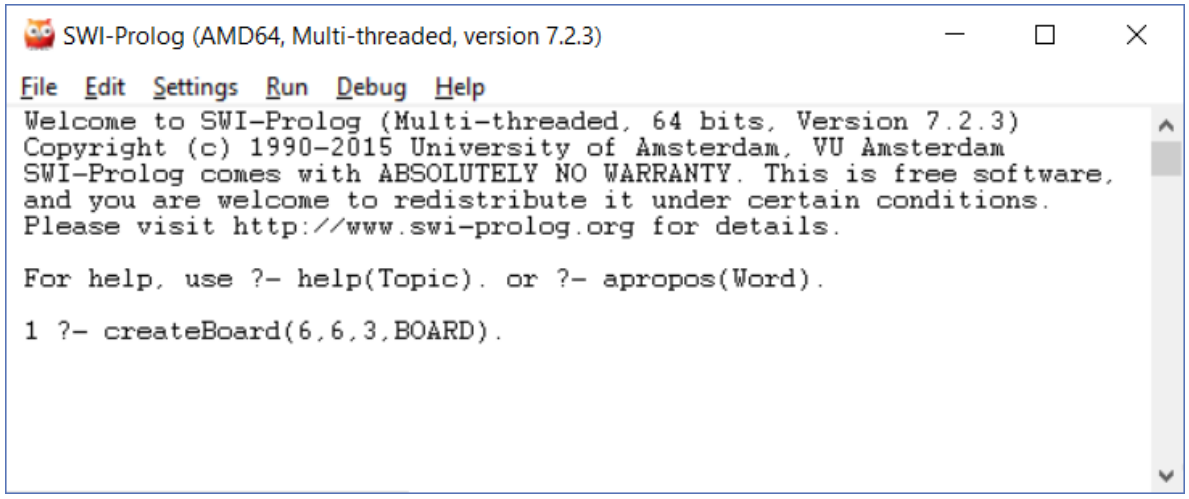

Figura 2-4: Consola SWI-Prolog

2.3 LINUX

Al igual que en Windows, el requisito principal es que Swi-Prolog esté instalado, sin embargo, el proceso de ejecución es un poco distinto, en LINUX no existe una ide como la que vimos anteriormente en Windows, sino que todo se realiza a través de consola. Sin embargo, para realizar una consulta basta con escribir: Consult(NombreArchivo.pl), reemplazar NombreArchivo por "battleship_189714319_MaturanaGuerra". Con esto ya podremos realizar nuestras consultas de forma normal al igual que en Windows.

CAPÍTULO 3. FUNCIONALIDADES OFRECIDAS

En este capítulo se expondrán las funciones principales, es decir, las que permiten el desarrollo de la partida.

Antes de comenzar es necesario aclarar a que este paradigma no posee variables como las conocemos y los predicados sólo responden si una consulta es verdadera o no, por lo tanto, para poder "retornar" algo (tablero, entero, etc) se hace a través de la misma consulta, dejando una "variable" la cual satisface de forma verdadera la consulta.

3. REQUERIMIENTOS FUNCIONALES OBLIGATORIAS.

1. **CreateBoard** (N, M, NumShips, BOARD): Predicado determina permite crear 3 tableros de dimensiones 6x6 (N y M respectivamente) con 3 barcos (NumShips) distintos, 2 tableros de dimensiones 10x10 con 5 barcos distintos, y 1 tablero de dimensión 20x20 con 10 barcos distintos (letra y dimensiones). Cada uno de los tableros es elegido de forma aleatoria (si existe más de un tipo de tablero).
2. **CheckBoard** (BOARD): Predicado que permite establecer la validez de un tablero BOARD, la función realiza las siguientes verificaciones:
    - Todas las listas de listas deben tener el mismo largo (cantidad de columnas)
    - Verifica todas y cada una de las celdas del tablero sean coherentes con la representación dada, esto quiere decir que la celda corresponda a su enumeración.

```
[[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 3-1: Tablero de 6x6 y su representación

- En caso de que una celda no siga el patrón de enumeración verifica que el elemento en la celda pertenezca a un barco (de la ***a*** a la ***j***), de no cumplir alguna de éstas tres condiciones el tablero es considerado inválido.

3. **Play** (BOARD, Ship, Posiciones): Predicado que recibe una jugada y determina si ésta es válida o no, puede ser sólo una posición o un conjunto de ellas. Las posiciones tienen la siguiente representación: [X1, Y1, X2, Y2, … XN, YN]. El predicado se encarga también de verificar que el disparo se efectúe en la mitad correspondiente al enemigo, de lo contrario la respuesta del predicado es FALSE.

3.1 FUNCIONES EXTRAS.

1. BoardToString (BOARD, BOARDSTR): Predicado que permite mostrar por pantalla un tablero dado (BOARD) permite mostrarlo por pantalla de forma que para el usuario sea entendible, como la representación es matricial (listas de listas) resulta provechoso ya que la mejor forma de mostrarla al usuario es como el tablero original del juego.

2. GetScore: Función que permite obtener el puntaje de una partida en curso o finalizada, el puntaje va desde 0 hasta un puntaje máximo. Esta función considera la cantidad de disparos exitosos realizados por el jugador, cantidad que el jugador ha destruido, la diferencia existente entre los barcos que posee el jugador y el enemigo, además de los disparos fallidos que el jugador ha efectuado. La fórmula es la siguiente.

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

Figura 3-2: Función getScore.

CAPÍTULO 4. RESULTADOS DE PRUEBAS

4. CREAR UN TABLERO DE 20 FILAS Y 20 COLUMNAS QUE SE ENCUENTRE EN LA BASE DE CONOCIMIENTOS.

El tablero fue creado con 5 barcos: El tablero será mostrado por pantalla mediante boardToString.

createBoard(20, 20, 10, BOARD).

```
1 ?- createBoard(20,20,10,BOARD),boardToString(BOARD,STR).
|a|  |  |  |  |j|  |  |  |  |  |  |  |  |a|a|a|
|a|  |  |  |  |j|  |  |  |  |  |c|b|b|  |  |  |
|a|  |  |  |  |j|  |  |  |  |  |d|  |  |  |  |  |
|  |  |  |  |  |j|  |  |  |  |  |d|  |  |  |  |  |
|  |  |  |  |  |j|  |  |  |  |  |d|  |  |  |  |  |
|  |b|b|  |c|  |j|  |  |  |  |  |d|  |  |  |  |  |
|d|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |j|
|d|  |  |  |  |  |  |  |  |e|e|e|e|e|  |  |  |j|
|d|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |j|
|d|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |j|
|  |e|e|e|e|e|  |  |  |  |  |  |  |  |  |  |  |j|
|  |  |i|i|i|i|i|  |  |  |  |  |f|f|f|f|f|f|  |
|h|h|h|h|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |g|  |  |  |  |  |  |i|i|i|i|i|  |  |h|  |
|f|  |  |g|  |  |  |  |  |  |  |  |  |  |  |  |h|  |
|f|  |  |g|  |  |  |  |  |  |  |  |  |  |  |  |h|  |
|f|  |  |g|  |  |  |  |  |  |  |  |  |  |  |  |h|  |
|f|  |  |g|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|f|  |  |g|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|f|  |  |g|  |  |  |  |  |  |g|g|g|g|g|g|g|  |  |
```

Figura 4-1: Tablero 20x20.

El tablero es creado de forma exitosa, posicionando los barcos tanto del jugador como del enemigo.

4.1 VERIFICAR LA VALIDEZ DE UN TABLERO MEDIANTE CHECKBOARD, EL CUAL PRESENTE PROBLEMAS.

En la base de conocimientos se proporcionan tableros que no poseen las características suficientes como para ser considerados válidos, como el que se muestra a continuación el

cual posee un barco con la letra “k” el cual no está registrado en la base de conocimientos como un barco admitido.

```
1 ?- tablero(2,10,10,5,BOARD),boardToString(BOARD,STR).

| | | | | | | |a|a|a|
| |d|d|d|d| | | | |b|
|c| | | | | | | |c|b|
| | | | | | | | | | |
| | |k|k| | |d|d|d|d|
| | | | | | | | | |e|
|e|e|e|e|e| | | | |e|
| |a|b|b| | | | | |e|
| |a| | | | | | | |e|
| |a| | | | |k|k| |e|
```

Figura 4-2: Tablero de 20x20, barco no admitido

```
2 ?- tablero(2,10,10,5,BOARD),checkBoard(BOARD).
false.
```

Figura 4-3: Consulta por la validez del tablero.

4.2 ATAQUE JUGADOR EN 3 POSICIONES ENEMIGAS.

Se atacará en el tablero de 20x20 (ver Fig. 4-1) en las posiciones {(0,17), (0,18), (0,19)}.

Por lo tanto la consulta debe ser de la siguiente forma:

Play(BOARD,a,[0,17, 0,18, 0,19].

Donde BOARD es el tablero en cuestión (Fig. 4-1)

```
    [f,362,363,g,365,366,367,368,369,370,371,372,373,374,375,376,377,378,379,380],
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |      [f,382,383,g,385,386,387,388,389,390,391,g,g,g,g,g,g,g,399,400]],a,[0,17,0,18,0,19]).
true .
```

Figura 4-4: Extracto de la consulta realizada ataques en 3 posiciones.

CAPÍTULO 5. POSIBLES ERRORES

Todas las pruebas realizadas fueron exitosas ya que en el diseño y la implementación fueron considerados todos y cada uno de éstos casos. Sin embargo, existe un caso que no fue considerado, el cual reside en la función checkBoard, ya que esta función no verifica que los barcos estén posicionados de forma yuxtapuestas todas sus unidades, es decir puede haber

barcos en diagonal e incluso barcos repartidos por todo el tablero, a pesar que con este error la partida puede darse sin mayores inconvenientes se quebranta una regla fundamental del juego, por lo que se recomienda utilizar los tableros que existen en la base de conocimientos.