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

MANUAL DE USUARIO

PROLOG

Nombre: Cristian Espinoza Silva
Profesor: Miguel Truffa Montenegro
Ayudante: René Zarate
Fecha de Entrega: 7 de noviembre

Santiago de Chile

2 - 2016

TABLA DE CONTENIDO

ÍNDICE DE FIGURAS

CAPÍTULO 1. INTRODUCCIÓN

En el siguiente documento se presenta una forma adecuada y cómoda de cómo utilizar la aplicación, dentro de lo que se procede a explicar, se encuentra el ¿Cómo compilar la aplicación ?, ¿Cómo utilizarla de manera adecuada?, ¿Qué funcionalidades cumple la aplicación ?, además de dar a conocer posibles errores que se pueden encontrar dentro de su ejecución.

La aplicación se desarrollará dentro del compilar – interprete *Swi.prolog versión 7.2.3,* se recomienda tener lenguaje técnico superior al nivel básico, para poder utilizar la aplicación de una manera adecuada y satisfactoria.

La aplicación a desarrollar es el juego "*Batalla Naval*", *en* esta oportunidad será en el paradigma lógico llevado a cabo en el lenguaje de programación *Prolog.*

Por último , se describe el llamado a los predicados de la aplicación para que puedan entregar los resultados que el usuario está solicitando.

CAPÍTULO 2. INSTRUCCIONES

Esta aplicación como se menciona anteriormente se llevara a cabo dentro del compilador – interprete *Swi-prolog versión 7.2.3*, que se puede descargar desde el siguiente link:

- http://www.swi-prolog.org/Download.html

Luego de haber terminado el paso anterior y haber instalado la aplicación, se procede a abrir la aplicación, obteniendo algo parecido a lo que muestra la figura 2-1:

Figura CAPÍTULO 2-1 : Campo principal de Swi-prolog

A continuación, se abre la aplicación creada este proceso se hace como lo muestra la figura 2-2:

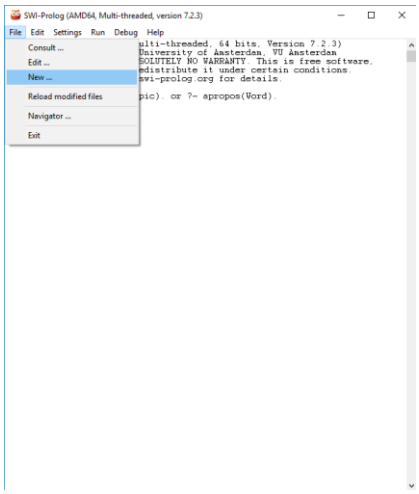

Figura CAPÍTULO 2-2: Representación de abrir un archivo.

De tal forma que al abrir la aplicación que contiene el código fuente, se podrá apreciar de la siguiente manera, como se muestra en la figura 2-3:

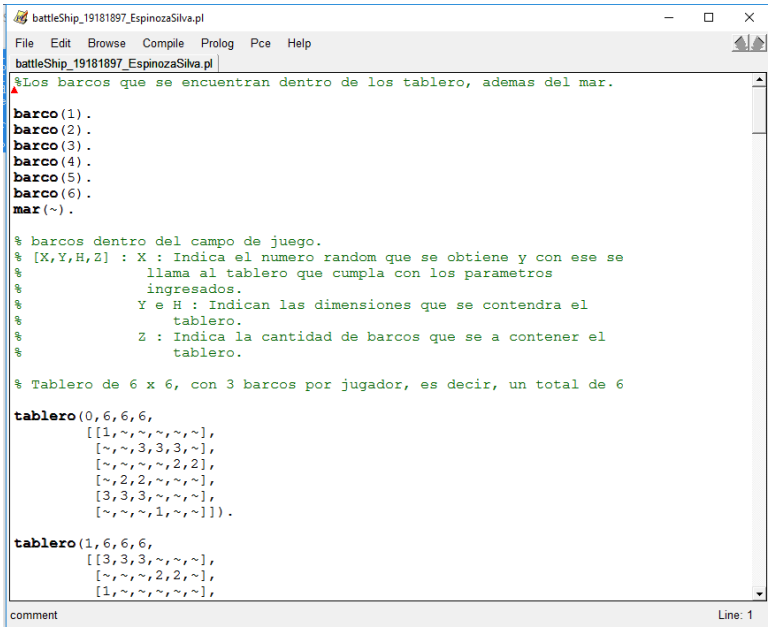

Figura CAPÍTULO 2-3: Representación de código fuente.

Luego para poder interpretar la aplicación se procede a realizar el siguiente paso que se verá mostrado en la figura 2-4:

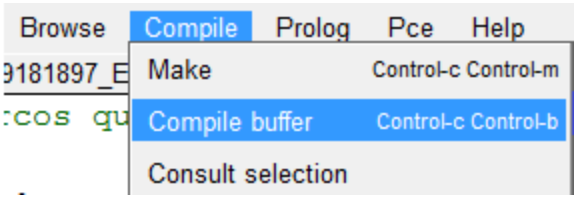

Figura CAPÍTULO 2- 4:Forma de interpretar la aplicación.

Al terminar los pasos anteriores la aplicación esta lista para poder hacer el llamado a los predicados y otorgarle los resultados que obtenga. Tener en cuenta que estos resultados se verán reflejados mediante la consola que facilita el intérprete *Swi-prolog.*

Algunas herramientas extras que nos entrega el intérprete *Swi-prolog*, es que permite ir viendo la *trace* de algún predicado, esto se puede ocupar como lo muestra la figura 2-5:

```
3 ?- trace.
true.

[trace] 3 ?- createBoard(6,6,6,BOARD).
   Call: (7) createBoard(6, 6, 6, _G503) ? creep
   Call: (8) _G582 is random(4) ? creep
   Exit: (8) 0 is random(4) ? creep
   Call: (8) tablero(0, 6, 6, 6, _G503) ? creep
   Exit: (8) tablero(0, 6, 6, 6, [[1, ~, ~, ~, ~, ~], [~, ~, 3, 3, 3,
 ~], [~, ~, ~, ~, 2|...], [~, 2, 2, ~|...], [3, 3, 3|...], [~, ~|...]
]) ? creep
   Exit: (7) createBoard(6, 6, 6, [[1, ~, ~, ~, ~, ~], [~, ~, 3, 3, 3
, ~], [~, ~, ~, ~, 2|...], [~, 2, 2, ~|...], [3, 3, 3|...], [~, ~|...
]]) ? creep
BOARD = [[1, ~, ~, ~, ~, ~], [~, ~, 3, 3, 3, ~], [~, ~, ~, ~, 2, 2],
[~, 2, 2, ~, ~|...], [3, 3, 3, ~|...], [~, ~, ~|...]] .
```

Figura CAPÍTULO 2-5: Instrucción para poder observar la trace de un predicado.

Para poder sacar este modo de interpretación, basta con hacer lo que muestra la figura 2-6:

```
[trace] 4 ?- notrace.
true.
```

Figura CAPÍTULO 2 - 6: Instrucción para sacar el modo trace.

CAPÍTULO 3. FUNCIONES

En la aplicación se desarrollaron la totalidad de funcionalidades mínimas, además de la funcionalidad *boardToString* que pertenece a las funcionalidades extras.

Se explicará cómo hacer el llamado a cada una de los predicados que contienen envuelta a la totalidad de las funcionalidades implementadas.

El predicado *createBoard,* recibe como parámetro: N, M, NumShip y BOARD que corresponden a:

- N y M: Dimensiones las cuales contendrá la matriz.
- NumShip: Cantidad de barcos que contiene la matriz.
- BOARD: Variable donde se almacena la matriz creada.

Se debe tener en consideración que el predicado *createBoard*, solo fue implementados para matrices previamente definidos que son los siguientes:

- Matriz de 6 x 6 – Contiene 3 *ship* por jugador.
- Matriz de 10 x 10 – Contiene 5 *ship* por jugador.
- Matriz de 20 x 20 – Contiene 10 *ship* por jugador.

Además, se debe tener en consideración que se implementaron tableros inválidos, un tablero para cada categoría.

A continuación, la figura 3-7 mostrara como hacer el llamado al predicado *createBoard* para que cree un tablero de 6 x 6:

```
9 ?- createBoard(6,6,6,BOARD),display(BOARD).
[[~,2,3,3,3,~],[~,2,~,~,~,~],[~,1,~,~,~,6],[~,~,~,2,2,~],[~,3,3,3,~,~
],[1,~,~,~,~,6]]
BOARD = [[~, 2, 3, 3, 3, ~], [~, 2, ~, ~, ~, ~], [~, 1, ~, ~, ~, 6],
[~, ~, ~, 2, 2|...], [~, 3, 3, 3|...], [1, ~, ~|...]] .

10 ?- createBoard(6,6,6,BOARD).
BOARD = [[1, ~, ~, ~, ~, ~], [~, ~, 3, 3, 3, ~], [~, ~, ~, ~, 2, 2],
[~, 2, 2, ~, ~|...], [3, 3, 3, ~|...], [~, ~, ~|...]]
```

Figura CAPÍTULO 3-7: Llamado a predicado createBoard.

El predicado *checkBoard* recibe como parámetro un *Board,* al cual se le realiza la comprobación si cumple con las reglas fundamentales del juego.

Se debe tener en consideración que, al implementar tableros inválidos dentro de la aplicación, existirán casos que el predicado *checkBoard,* otorgara false.

El llamado se hace como lo muestra la figura 3-8 y 3-9:

```
12 ?- createBoard(6,6,6,BOARD),checkBoard(BOARD).
BOARD = [[1, ~, ~, ~, ~, ~], [~, ~, 3, 3, 3, ~], [~, ~, ~, ~, 2, 2],
[~, 2, 2, ~, ~|...], [3, 3, 3, ~|...], [~, ~, ~|...]]
```

Figura CAPÍTULO 3-8: Llamado exitoso ha predicado checkBoard.

```
11 ?- createBoard(6,6,6,BOARD),checkBoard(BOARD).
false.
```

Figura CAPÍTULO 3-9: Llamado fallido ha predicado checkBoard.

El predicado *play* recibe como parámetro *Board*, *Ship* y Posiciones que corresponden a:

- *Board:* Variable la cual contiene la matriz que se está utilizando en el juego.
- *Ship:* Barco con el cual el usuario desea realizar el ataque.
- Posiciones: Lista de coordenadas donde el usuario desea realizar el ataque.

El llamado al predicado *play*, se hace como lo muestra la imagen 3-10 y 3-11:

```
13 ?- createBoard(6,6,6,BOARD),play(BOARD,1,[1,2]).
jugada valida
BOARD = [[1, ~, ~, ~, ~, ~], [~, ~, 2, 2, ~, ~], [3, 3, 3, ~, ~, ~],
[~, 2, 2, ~, ~|...], [3, 3, 3, ~|...], [~, ~, ~|...]]
```

Figura CAPÍTULO 3-10: Llamado ha predicado play exitoso.

```
14 ?- createBoard(6,6,6,BOARD),play(BOARD,9,[1,2]).
false.
```

Figura CAPÍTULO 3-11: Llamado ha predicado play fallido.

El predicado *boardToString* recibe como parámetro *Board* y BoardStr que corresponden a:

- Board: Matriz que se esta ocupando en la partida y se quiere transformar a un tipo de dato string.
- BoardStr: Variable donde se va almacenar la matriz en tipo de dato string.

El llamado al predicado *boardToString*, se hace como lo muestra la imagen 3-12:

```
17 ?- createBoard(6,6,6,BOARD),boardToString(BOARD,BoardStr).
BOARD = [[~, 2, 3, 3, 3, ~], [~, 2, ~, ~, ~, ~], [~, 1, ~, ~, ~, 6],
[~, ~, ~, 2, 2|...], [~, 3, 3, 3|...], [1, ~, ~|...]],
BoardStr = "1,~,~,~,~,6,~,3,3,3,~,~,~,~,~,2,2,~,~,1,~,~,~,6,~,2,~,~,~
,~,~,2,3,3,3,~"
```

Figura CAPÍTULO 3-12: Llamado ha predicado boardToString.

CAPÍTULO 4. POSIBLES ERRORES

La aplicación fue pensada y creada para que funcione con ciertos parámetros de entrada, por lo que, si el usuario introduce unos parámetros que el predicado no está esperando existe una gran probabilidad que el predicado entregue algún resultado no esperado o hasta un posible error.

A continuación, se describirán algunos de los posibles errores que puede tener la aplicación:

- Si introduce algún número no entero cuando desea realizar verificar un ataque, el compilador – interprete *Swi-prolog* entregara un error, ya que el predicado está esperando otro tipo de entrada. El error se puede aprecia de mejor forma en la figura 4-13:

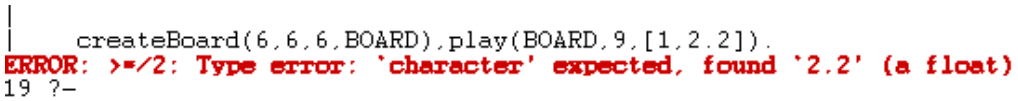

Figura CAPÍTULO 4-13: Error de entrega de parámetros.

- En el caso que la aplicación reciba mucha información, puede resultar falta de memoria por parte de compilar – interprete *Swi-prolog* y por parte del sistema. Si esto llegase a ocurrir puede que la aplicación no entregue los resultados deseados.