

Paradigmas de programación. Laboratorio 2: Scheme.

**Andrés Felipe Muñoz Bravo
19.646.487-5**

Profesor:
Roberto González Ibáñez

Santiago - Chile
2016

TABLA DE CONTENIDOS

Tabla de Contenidos.....	I
Índice de Figuras	I
CAPÍTULO 1. Introducción	2
1.1 Descripción del problema.....	2
1.2 Objetivos	2
1.3 herramientas utilizadas.....	3
CAPÍTULO 2. Descripción de la solución	3
2.1 Tipos de datos Abstractos	3
2.1.1 TDA Position:	3
2.1.2 TDA Matriz:	4
2.1.3 TDA Board:.....	4
2.2 Funciones	5
2.2.1 createBoardRC y createBoardRL:.....	5
2.2.2 CheckBoard:.....	6
2.2.3 PutShip:	7
2.2.4 Play:.....	7
2.2.5 Board a String:	9
2.2.6 getScore.....	9
CAPÍTULO 3. Análisis de los resultados	10
CAPÍTULO 4. Conclusión.....	11
CAPÍTULO 5. Referencias	11

ÍNDICE DE FIGURAS

Ilustración 1: TDA Position.	3
Ilustración 2: Ejemplo 1 TDA Matriz.	4
Ilustración 3: Ejemplo 2 TDA Matriz	4
Ilustración 4: TDA Board.....	5
Ilustración 5:checkBoard, salida verdadera.....	6
Ilustración 6:checkBoard, salida falso	6
Ilustración 7: PutShip doble.	7
Ilustración 8: Código función play.....	8

CAPÍTULO 1. INTRODUCCIÓN

1.1 DESCRIPCION DEL PROBLEMA

Se ha pedido realizar un juego basado en otro juego llamado “Battle ship”, su traducción literal es “Batalla Naval”. Este juego necesita dos jugadores, uno será la computadora y el otro será un usuario. El juego transcurre a través de turnos en los cuales cada jugador, realiza un ataque al oponente, el cual puede resultar en la destrucción de un barco. El juego finaliza cuando uno de los dos contrincantes no posee barcos.

1.2 OBJETIVOS

Los objetivos del proyecto son, realizar un estudio sobre el problema, para poder llegar a una solución óptima, la cual será implementada posteriormente. Para ello se ha dividido este gran problema en subproblemas específicos a realizar, estos se mencionan a continuación:

1. Crear TDA Position
2. Crear TDA Matriz
3. Crear TDA Board
4. Utilizar los tipos de datos abstractos anteriores para poder realizar las siguientes funciones que operan sobre el TDA Board:
 - a. Crear tablero.
 - b. Verificar si el tablero es válido.
 - c. Ubicar barcos del enemigo en el tablero.
 - d. Ubicar barcos del usuario en el tablero.
 - e. Implementar una jugada sobre el tablero.
 - f. Transformar el tablero a string.

1.3 HERRAMIENTAS UTILIZADAS

Para desarrollar la aplicación, se utiliza el lenguaje de programación Scheme, que opera bajo el paradigma de programación Funcional. Este paradigma está basado en las funciones matemáticas, donde no existen efectos colaterales, solo se transforman las entradas en una salida. Cabe recalcar que no existen variables que van mutando a través del transcurso del programa.

El intérprete utilizado para este lenguaje es DrRacket en la versión 6.6. Además el programa se rige por el estándar R6RS.

CAPÍTULO 2. DESCRIPCION DE LA SOLUCIÓN

2.1 TIPOS DE DATOS ABSTRACTOS

El desarrollo de la aplicación está basado en tres tipos de datos abstractos (TDA); Position, Matriz, Board. Estos tipos de datos abstractos implementados serán explicados detalladamente a continuación.

2.1.1 TDA Position:

Este tipo de dato implementado, es utilizado para indicar posiciones dentro de una matriz, está conformado por un par, donde el primer elemento indica la fila y el segundo elemento la columna.

```
> (createPosition 2 3)
{2 . 3}
```

Ilustración 1: TDA Position.

2.1.2 TDA Matriz:

Este tipo de dato implementado, hace alusión a una matriz en la cual estarán posicionados los barcos de los contrincantes. Esta matriz está conformada por lista de listas que contienen solamente caracteres. Estos caracteres representan los estados de una casilla, los cuales pueden ser que exista; un barco, que haya sido atacado, que sea agua (no atacado).

Los constructores del tipo de dato Matriz son dos, uno que utiliza recursión lineal y otro que utiliza recursión de cola.

```
> (createMatrizRL 10 10)
{{#\. #\. #\. #\. #\. #\. #\. #\. #\. #\.}
 {#\. #\. #\. #\. #\. #\. #\. #\. #\. #\.}
 {#\. #\. #\. #\. #\. #\. #\. #\. #\. #\.}
 {#\. #\. #\. #\. #\. #\. #\. #\. #\. #\.}
 {#\. #\. #\. #\. #\. #\. #\. #\. #\. #\.}
 {#\. #\. #\. #\. #\. #\. #\. #\. #\. #\.}
 {#\. #\. #\. #\. #\. #\. #\. #\. #\. #\.}
 {#\. #\. #\. #\. #\. #\. #\. #\. #\. #\.}
 {#\. #\. #\. #\. #\. #\. #\. #\. #\. #\.}
 {#\. #\. #\. #\. #\. #\. #\. #\. #\. #\.}}
```

Ilustración 2: Ejemplo 1 TDA Matriz.

```
> (createMatrizRC 10 10)
{{#\. #\. #\. #\. #\. #\. #\. #\. #\. #\.}
 {#\. #\. #\. #\. #\. #\. #\. #\. #\. #\.}
 {#\. #\. #\. #\. #\. #\. #\. #\. #\. #\.}
 {#\. #\. #\. #\. #\. #\. #\. #\. #\. #\.}
 {#\. #\. #\. #\. #\. #\. #\. #\. #\. #\.}
 {#\. #\. #\. #\. #\. #\. #\. #\. #\. #\.}
 {#\. #\. #\. #\. #\. #\. #\. #\. #\. #\.}
 {#\. #\. #\. #\. #\. #\. #\. #\. #\. #\.}
 {#\. #\. #\. #\. #\. #\. #\. #\. #\. #\.}
 {#\. #\. #\. #\. #\. #\. #\. #\. #\. #\.}}
```

Ilustración 3: Ejemplo 2 TDA Matriz

2.1.3 TDA Board:

Este tipo de dato implementado, refiere al tablero del juego, esta formado por una lista de varios elementos especificados a continuación:

- Posición 0: Cantidad de barcos del computador
- Posición 1: Cantidad de barcos del usuario (jugador).
- Posición 2: Puntaje obtenido por la computadora.
- Posición 3: Puntaje obtenido por el usuario (jugador).
- Posición 4: Matriz que contiene los barcos de los jugadores.
- Posición 5: Lista con historial de jugadas (aciertos o no).

```

> (createBoardRC 10 10 10 10)
{10 Posición 0
 0 Posición 1
 0 Posición 2
 0 Posición 3
 {{#\ . #\E #\ . #\ . #\ . #\ . #\ . #\E #\ .}
 {#\ . #\ . #\ . #\ . #\E #\ . #\ . #\E #\ . #\ .}
 {#\E #\ . #\ . #\ . #\ . #\E #\ . #\ . #\ . #\ .}
 {#\ . #\E #\ . #\ . #\E #\ . #\ . #\ . #\ . #\ .}
 {#\ . #\ . #\E #\ . #\ . #\ . #\ . #\E #\ . #\ .}
 {#\ . #\ . #\ . #\ . #\ . #\ . #\ . #\ . #\ . #\ .}
 {#\ . #\ . #\ . #\ . #\ . #\ . #\ . #\ . #\ . #\ .}
 {#\ . #\ . #\ . #\ . #\ . #\ . #\ . #\ . #\ . #\ .}
 {#\ . #\ . #\ . #\ . #\ . #\ . #\ . #\ . #\ . #\ .}
 {#\ . #\ . #\ . #\ . #\ . #\ . #\ . #\ . #\ . #\ .}
 () } Posición 5

```




Ilustración 4: TDA Board.

2.2 FUNCIONES

Para el manejo del juego existen diferentes funciones, las cuales se presenta detalladamente a continuación, cabe recalcar que en ellas subyacen otras funciones.

2.2.1 createBoardRC y createBoardRL:

Existen una serie de funciones que realizan pasos previos para crear un tablero, estos se mencionan a continuación:

1. Crear una matriz base con caracteres iguales (#\.) que representan agua (no ha sido atacado). Esta matriz se puede crear según recursión lineal como también por recursión de cola, por lo tanto para createBoarRC se utiliza la funcion que crea una matriz con recursión de cola y para createBoardRL se utiliza la funcion que crea una matriz a través de recursión lineal.
2. Crear una lista de posiciones aleatorias validas dentro de la matriz.
3. Ubicar en las la matriz un carácter (barco “#\E) en cada posición aleatoria generada.
4. Crear una lista con los elementos anteriormente señalados (ver punto 2.1.3).

2.2.2 CheckBoard:

Esta función permite verificar si es un tablero valido para el desarrollo del juego. Para eso se realizan las siguientes verificaciones:

- Verificar si existe la cantidad correspondiente de elementos en la lista board.
- Verificar que los primeros 4 elementos son números
- Verificar que el quinto elemento es una matriz:
 - Verificar que la matriz es una lista de listas
 - Verificar que el número de filas es par
 - Verificar que tiene el mismo número de columnas en todas las filas.
 - Verificar que cada elemento de la matriz sea un carácter.
- Verificar que el sexto elemento es una lista

Una vez verificada todas estas condiciones anteriores la función checkBoard retorna el valor booleano verdadero (#t).

```
> (checkBoard boardRC)
#t
```

Ilustración 5:checkBoard, salida verdadera

```
> (checkBoard (list (list 2 3) (list 2 3) (list 6 2 4)))
#f
```

Ilustración 6:checkBoard, salida falso

2.2.3 PutShip:

Esta función retorna un nuevo tablero agregando un barco en la posición entregada, para eso se necesita realizar las siguientes acciones.

- Verificar que el tablero entregado es un tablero (con checkBoard).
- Verificar que la posición entregada es una posición (pertenencia del TDA Position).
- Verificar que la posición entregada es válida, esto quiere decir que no sea una posición fuera del tablero.
- Cumplidas todas las condiciones anteriores se procede a hacer dos modificaciones en el tablero con ayuda de los modificadores del TDA Board. Estas modificaciones son:
 - Cambiar el carácter en la matriz del board en la posición indicada, con ayuda del modificador perteneciente al TDA Matriz.
 - Cambiar la cantidad de barcos que tiene el usuario.

```
> (putShip(putShip boardRC (createPosition 0 1) #\M) (createPosition 1 3) #\M)
{4 2 0 0 {{#\E #\ . #\E #\ .} {#\ . #\E #\ . #\E} {#\ . #\M #\ . #\ .} {#\ . #\ . #\ . #\M}} {}}
```

Ilustración 7: PutShip doble.

2.2.4 Play:

Para hacer una jugada en el juego es necesario realizar dos subproblemas detallados cada uno a continuación:

- Jugada del usuario:
 - Verificar si la lista de posiciones entregada para el ataque del usuario es vacía (en caso de ser un disparo masivo la lista contendría más de un elemento). Esto es el caso base de la función debido a que es recursiva.

- Verificar si se puede hacer una jugada, esto se refiere a verificar si los jugadores contienen barcos dentro de su parte del tablero.
- Verificar la primera posición de la lista es válida para hacer la jugada.
- Hacer el llamado recursivo de esta función (jugada del usuario), con los cambios del tablero realizados según la posición atacada, entregando solo la cola de la lista de posiciones.
- Jugada de la computadora:
 - Verificar si se puede hacer una jugada, esto se refiere a verificar si los jugadores contienen barcos dentro de su parte del tablero. Cabe mencionar que esta función no es recursiva debido a que no es necesario recorrer una lista.
 - Generar una posición random la cual será atacada en la parte del tablero del usuario.
 - Entregar el tablero con el ataque realizado, cambiando el carácter y cantidad de barcos del tablero según corresponda, ya que puede o no haber dado en un objetivo.

Luego de tener las dos funciones anteriormente mencionadas, se procede a utilizarlas dentro de la función play de la siguiente forma:

```
(define play
  (lambda (board ship positions seed)
    (if (and (checkBoard board) (list? positions) (number? seed))
        (play-cpu (play-player board positions ship) seed)
        board)
  )
)
```

Ilustración 8: Código función play.

2.2.5 Board a String:

Esta función crea un string a partir de un tablero. Según los parámetros de entrada se mostraran o no los barcos del enemigo. Para poder implementar esta función, se dividió 2 subproblemas, y luego uno de ellos fue dividido en dos más como se muestra continuación.

1. Crear string de todo el talero. Esto se realiza concatenando cada fila de la matriz con un carácter de salto de línea entre ellas.
2. Crear string con los barcos enemigos escondidos.
 - Crear string de la mitad enemiga escondiendo los barcos. Esto se realiza concatenando cada carácter de la fila, en caso de que sea un barco se concatena un carácter que representa agua (#\.).Luego se concatena un salto de línea al final de cada fila.
 - Crear string con la mitad del usuario. Esto se realiza concatenando cada fila de la matriz con un carácter de salto de línea entre ellas.

Luego para el segundo subproblema solo basta concatenar los dos string formados en los subproblemas dentro de él.

2.2.6 getScore

Esta función retorna el puntaje obtenido del usuario durante el juego.

La implementación de esta funcion es simple debido a el TDA board implementado, donde se guarda el puntaje de usuario en la cuarta posición de la lista que compone a board. Para ello utilizamos un selector del TDA Board, obteniendo fácilmente el puntaje. Esto es posible debido a que cuando se hace una jugada se actualiza el puntaje.

CAPÍTULO 3. ANÁLISIS DE LOS RESULTADOS

Los resultados obtenidos para cada función son los esperados a continuación se detalla más cada uno de ellos.

- Implementación de los tipos de datos abstractos:
 - TDA Position: Se implementó de forma correcta según todas las capas que debe contener un TDA.
 - TDA Matriz: Se implementó de forma correcta según todas las capas que debe contener un TDA.
 - TDA Board: Se implementó de forma correcta según todas las capas que debe contener un TDA.
- La función para crear tablero funciona siempre en todas las pruebas realizadas.
- La función checkBoard es lo suficientemente robusta para verificar si es un tablero valido, de todas las pruebas realizadas todas funcionaron correctamente.
- La función putShip, para ubicar barcos también es lo suficientemente robusta para verificar que donde se está poniendo un barco es correcto o no. De las pruebas realizadas el 100% de ellas aprobó el test.
- La función play funciona perfectamente en todas las pruebas realizadas, lo único fue una modificación que no permitía jugar si no existían barcos de alguno de los dos contrincantes, esta modificación se realizó solo para efectos de que la evaluación del mismo sea más fácil.
- La función que convierte de tablero a string también funciona correctamente en todas las pruebas realizadas.
- La función getScore funciona al 100% cabe recalcar que para hacer estas pruebas fue necesario realizar alguna jugada que diera en algún barco, para así modificar el puntaje de algún jugador y poder ver su utilidad.

CAPÍTULO 4. CONCLUSIÓN

Todos los objetivos obligatorios fueron realizados con éxito. Además también se realizó de un objetivo extra.

Al principio fue muy complicado trabajar con scheme, ya que al cambiar de paradigma donde solo se utilizan recursiones como ciclos es bastante chocante. Además de no tener variables que modifican su estado con las cuales trabajar. Poco a poco se estudió primero el lenguaje, haciendo ejercicios propuestos en clases, consultando a compañeros, buscando videos en youtube, etc. Una vez entendido de mejor forma el paradigma y lenguaje de programación, todo fue más sencillo. Incluso fue mejor entendido y más fácil que el paradigma de programación Imperativo en el lenguaje de programación C.

La utilización de los tipos de datos abstractos, con un buen análisis, fue lo más importante para poder avanzar rápidamente, ya que una vez definidas todas las capas de estos (constructor, pertenencia, selectores, modificadores, otros), era más fácil trabajar con las ideas obtenidas en el análisis fue muy estudiado previamente.

Lo más difícil fue la utilización de recursiones, ya que la utilización de estas fue muy vagas en otros lenguajes de programación, incluso nulas. Hay que mencionar que a medida de que se iba avanzando en el desarrollo de la aplicación las recursiones fueron vista de forma mas natural.

En general scheme es un gran lenguaje de programación, y en parte se hace más sencillo separar los problemas en subproblemas, ya que se está obligado a entregar solamente un dato como retorno.

CAPÍTULO 5. REFERENCIAS

Departamento de Ingeniería Informática USACH. (2016). Enunciado proyecto. 2016: USACH.

Ulloa, P. (2/2016). Manual de bolsillo: Programación funcional scheme. Chile