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:
Giovanni Benussi
Mauricio Rojas
Esteban Contardo

Santiago - Chile
2-2016

TABLA DE CONTENIDOS

Tabla de Contenidos..... I
Índice de Figuras..... I
CAPÍTULO 1. introducción..... 1
1.1 Reglas..... 1
1.2 Cómo jugar..... 2
CAPÍTULO 2. Procesos de Compilación..... 3
2.1 Cómo compilar y ejecutar..... 3
2.2 Linux y Windows..... 3
CAPÍTULO 3. Funcionalidades ofrecidas..... 4
3. Funciones obligatorias. 5
3.1 Funciones Extras. 6
CAPÍTULO 4. Resultados de pruebas..... 6
4. Crear un tablero de 5 filas y 10 columnas. 6
4.1 Posicionamiento de 6 barcos del jugador en el tablero, 1 de ellos en la mitad del enemigo. 7
4.2 Ataque jugador en Fila y respuesta de cpu con ataque y posición aleatoria. 8
4.3 Atacar en el propio tablero. 8
CAPÍTULO 5. Posibles errores..... 9

ÍNDICE DE FIGURAS

Figura 2-1: DrRacket en el escritorio..... 3
Figura 2-2: Ubicación Archivo “.rtk”..... 3
Figura 2-3: Correr código. 4
Figura 3-1: Función getScore..... 6
Figura 4-1: Tablero 5x10..... 7
Figura 4-2: Posicionamiento barcos jugador. 7
Figura 4-3: Prueba ShowComplete 0..... 8
Figura 4-4: Prueba ShowComplete 1..... 8
Figura 4-5: Mismo Tablero..... 9

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, en esta implementación un jugador puede tener un barco de diferencia con respecto al otro, con 1 barco como mínimo, a diferencia de su implementación original donde cada jugador debe tener la misma cantidad de barcos.

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 **Racket** de amplio espectro de la familia **Lisp** y **Scheme**, a través de la herramienta DrRacket la cual es una herramienta de Entorno de Desarrollo Integrado (IDE). La plataforma se adhiere a la iniciativa de *Software Libre.* El proyecto fue realizado con la versión 6.3, sin embargo no existen problemas con otras versiones ya que se realizó bajo **R6RS** sugerido por la coordinación.

Para poder ejecutar el programa es necesario tener Drracket instalado, y tener el archivo "battleShip_189714319_MaturanaGuerra.rkt" el cual está adjunto al informe.

2.2 LINUX Y WINDOWS

Las imágenes expuestas a continuación se consiguieron en ambiente Linux, ambiente donde fue creado. Sin los pasos a seguir son idénticos en Windows.

Buscamos Drracket en nuestro ordenador y lo ejecutamos.

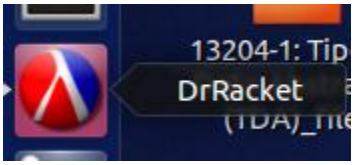

Figura 2-1: DrRacket en el escritorio

Una vez tengamos abierto Drracket, apretamos la combinación **ctrl+o** para abrir un archivo y buscamos la ubicación de nuestro archivo con extensión ".rkt" .

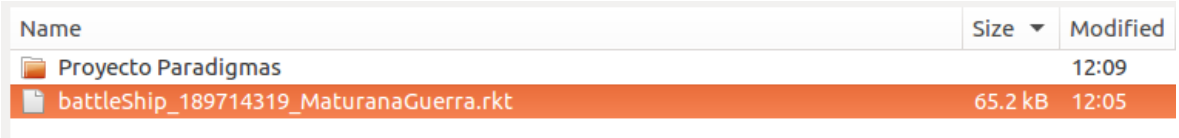

Figura 2-2: Ubicación Archivo ".rtk"

Una vez abierto el código debemos “correrlo” para probar las instrucciones que se encuentren en él, al final del código se adjunta una sección de pruebas. En la esquina superior derecha se encuentra un botón con la etiqueta “RUN”, al presionarlo se ejecutarán las instrucciones en él.

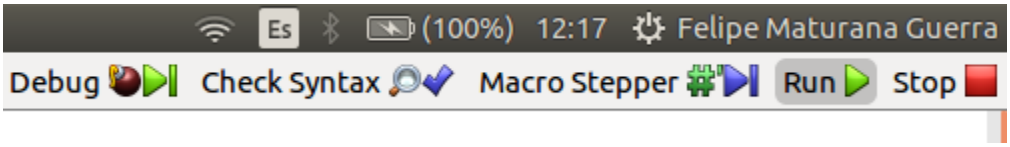

Figura 2-3: Correr código.

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 qué hace referencia cada argumento de las funciones que se expondrán a continuación y la forma que éstas poseen.

Cuando el parámetro diga:

- Ships: Hace referencia al TDA Ships, el cual consiste en una lista donde el primer elemento es la cantidad de barcos que posee ese TDA y los N elementos que lo acompañan son listas con las características de los N barcos donde N es igual a la cantidad señalada en el primer elemento, de la forma:

  ‘(2 (3 1 3 0 #\a 2) (3 1 3 1 #\b 2))

Cada barco posee sus características principales representadas de la siguiente forma: El **primer** elemento corresponde al **largo**, el **segundo** elemento hace referencia al **ancho** del barco, el **tercer** elemento corresponde a la **vida** del barco (largo * ancho), el **cuarto** elemento hace referencia a el **tipo de ataque** que posee el barco (0 normal, 1 para ataque en columna), el **quinto** elemento hace referencia al **carácter** que representa al barco (debe ser único) y el **último** elemento hace referencia al **comandante**, es decir, quién le pertenece el barco (1 para jugador, 2 para enemigo).

- Board: Hace referencia al TDA Battleship implementado, el cual posee todas las características necesarias para un correcto desarrollo de la partida,

- Seed: Es una semilla representada como un entero, cada entero distinto generará nuevos números aleatorios.
- Positions: Es una lista la cual en su primer elemento contiene un entero que señala cuántas posiciones posee, y a continuación una lista con todas las posiciones, una tras otra, de la forma: ‘(N (X1 Y1 X2 Y2 X3 Y3)), con N=3

Las funciones que permiten el desarrollo de una partida son las siguientes:

3. FUNCIONES OBLIGATORIAS.

1. CreateBoard RL o RC: Ambas realizan la misma función sólo cambia su implementación la cual es desconocida para el usuario, sin embargo para el programador difiere en el tipo de recursión que se utiliza para concatenar las listas. Recursión Lineal y de Cola respectivamente. Esta función recibe como argumento:
    a. N y M, enteros que representan la cantidad de Fila y Columna del tablero a crear, M debe ser par.
    b. Ships.
    c. Seed.

    Lo anteriormente descrito corresponde al conjunto de salida (dominio) su conjunto de llegada es: Listas. En especial, una que represente el TDA Battleship. Si algo sale mal será null.
2. CheckBoard: Función que verifica si un TDA Battleship cumple con las condiciones para ser considerado válido, a pesar que se verifican los elementos presentes en el TDA, se hace un especial énfasis a las condiciones del tablero presentes en él (7mo elemento del TDA). Recibe como parámetro un TDA Battleship) y el conjunto de llegada son los **booleanos**.
3. PutShip: Función que recibe un **TDA Battleship, TDA Ships**, una **posición** la cual debe tener sólo una coordenada, el programa se encargará de posicionarlo de acuerdo a las características del tablero, siempre intentará posicionarlo vertical, de no poder intentará posicionarlo horizontal, el conjunto de llegada son las listas, en especial las que representa el TDA Battleship.

4. Play: Función que permite realizar una jugada en el tablero, y recibe como respuesta el ataque de la CPU, en una posición pseudo-aleatoria y con un barco elegido de forma pseudo-aleatorio, además de registrar en el TDA Battleship el estado del último disparo, el último jugador, y si la partida ha finalizado cuando corresponda hacerlo, es decir cuando la cantidad de barcos del jugador o del enemigo llegue a cero. El conjunto de llegada de esta función es: Listas, en especial las que representan al TDA Battleship. Si algo salió mal es null.
5. Board -> string: Función que recibe un **TDA Battleship** como parámetro y obtiene el tablero para luego imprimirlo de tal forma que sea legible y entendible para el usuario.

3.1 FUNCIONES EXTRAS.

1. 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.

$$score = (DE * 100 + BD * 400 + (BJ - BE) * 200) - DF * 50,$$

$$if\ score < 0 \rightarrow score = 0$$

Figura 3-1: Función getScore

CAPÍTULO 4. RESULTADOS DE PRUEBAS

Las siguientes pruebas se realizarán con una semilla con valor = 1 y se mostrarán por pantalla mediante la función board -> string con show complete 1. .

4. CREAR UN TABLERO DE 5 FILAS Y 10 COLUMNAS.

El tablero fue creado con 4 barcos:

(createBoardRC 5 10 (Ships 4 '(1 #\a 1 #\b 2 #\c 2 #\d)) 1)

Figura 4-1: Tablero 5x10

4.1 POSICIONAMIENTO DE 6 BARCOS DEL JUGADOR EN EL TABLERO, 1 DE ELLOS EN LA MITAD DEL ENEMIGO.

Los barcos se intentarán posicionar en las posiciones #\e (0,1) #\c (1,0) #\d (1,1) #\a (4,1) #\c (1,2) #\f (4,7). Cabe mencionar que la notación de las posiciones utilizadas es de acuerdo a la notación propia del TDA el cuál comienza del 0, en notación matricial el 1,1, corresponde al (0,0) del TDA.

(board->string (putship (putship (putship (putship (putship (putship (createBoardRC 5 10 (Ships 4 '(1 #\a 1 #\b 2 #\c 2 #\d)) 1) (list 1 (list 1 1)) '(3 1 3 1 #\d 1)) (list 1 (list 0 1)) '(3 1 3 1 #\e 1)) (list 1 (list 4 1)) '(3 1 3 1 #\a 1)) (list 1 (list 1 0)) '(3 1 3 1 #\b 1)) (list 1 (list 1 2)) '(3 1 3 1 #\c 1)) (list 1 (list 4 7)) '(3 1 3 1 #\f 1)) 1)

Figura 4-2: Posicionamiento barcos jugador.

Sólo se posicionan los 5 primeros barcos, b d y c se posicionan sin problemas, sin embargo e no podía posicionarse de forma vertical por lo cual el algoritmo prueba un posicionamiento horizontal el cuál si podía efectuarse ya que éste posee celdas libres que pertenezcan a la mitad del jugador, caso idéntico a lo que sucede con el barco #\a.

4.2 ATAQUE JUGADOR EN FILA Y RESPUESTA DE CPU CON ATAQUE Y POSICIÓN ALEATORIA.

El mismo tablero anterior es recibido como parámetro de entrada para el ataque del jugador en la primera Fila del tablero del enemigo, como respuesta la cpu eligiera una posición aleatoria y de acuerdo a su disposición armamentística se realizar un ataque especial o no.

(board->string (play (putship (putship (putship (putship (putship (createBoardRC 5 10 (Ships 4 '(1 #\a 1 #\b 2 #\c 2 #\d)) 1) (list 1 (list 1 1)) '(3 1 3 1 #\d 1)) (list 1 (list 0 1)) '(3 1 3 1 #\e 1)) (list 1 (list 4 1)) '(3 1 3 1 #\a 1)) (list 1 (list 1 0)) '(3 1 3 1 #\b 1)) (list 1 (list 1 2)) '(3 1 3 1 #\c 1)) '(1 (3 1 3 1 #\d 1)) (list 5 (list 0 5 0 6 0 7 0 8 0 9)) 10) 1)

```
|.|0|e|e|.|@|0|0|X|@|
|b|0|c|.|.|.|.|.|.|.|
|b|0|c|.|.|.|.|.|.|.|
|b|X|c|.|.|.|.|.|.|.|
|.|0|a|a|.|.|.|.|.|.|
```

Figura 4-3: Prueba ShowComplete 0

```
|.|0|e|e|.|@|0|0|X|@|
|b|0|c|.|.|.|.|d|d|d|
|b|0|c|.|.|.|b|b|b|.|
|b|X|c|.|.|.|.|.|.|.|
|.|0|a|a|.|c|c|c|.|.|
```

Figura 4-4: Prueba ShowComplete 1

Como se aprecia en la imagen anterior, el enemigo poseía en su armamento disparos en columna, como se aprecia en las figuras 4-3 y 4-4, con el parámetro showcomplete 0 y 1.

4.3 ATACAR EN EL PROPIO TABLERO.

En la siguiente prueba se hará la prueba de realizar un dispararo inválido

(board->string (play (putship (putship (putship (putship (putship (createBoardRC 5 10 (Ships 4 '(1 #\a 1 #\b 2 #\c 2 #\d)) 1) (list 1 (list 1 1)) '(3 1 3 1 #\d 1)) (list 1 (list 0 1)) '(3 1 3 1 #\e 1)) (list 1 (list 4 1)) '(3 1 3 1 #\a 1)) (list 1 (list 1 0)) '(3 1 3 1 #\b 1)) (list 1 (list 1 2)) '(3 1 3 1 #\c 1)) '(1 (3 1 3 1 #\d 1)) (list 5 (list 0 0 0 1 0 2 0 3 0 4)) 10) 1)

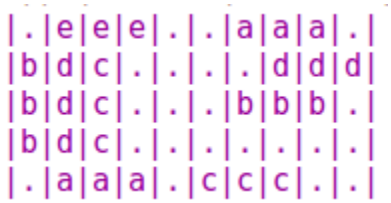

Figura 4-5: Mismo Tablero

Como resultado la función retorna el mismo tablero.

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. En caso de fallos es necesario verificar, en primera instancia, el compilador que se está utilizando ya que a pesar de haber seguido todos los parámetros de estandarización y haber utilizado sólo las librerías estándares según R6RS

Existe una potencial falla la cual consiste en posicionar dos barcos con la misma letra, en primera instancia no habrá problemas, pero al momento que un barco muera, se darán los dos por muertos, es por esto que es necesario que tanto el jugador como la cpu posean barcos con letras únicas, este problema puede ser evitado a través de una verificación al momento de posicionar los barcos del jugador, preguntar si la letra del barco a posicionar ya existe, y al momento de crear el tablero verificar que todos los barcos posean letras únicas.