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

Paradigmas de programación

Manual de usuario

Cristian Eduardo Espinoza Silva

Profesor:
MIGUEL TRUFFA MONTENEGRO

Santiago - Chile
2016

TABLA DE CONTENIDOS

TABLA DE FIGURAS

CAPÍTULO 1. INTRODUCCIÓN

El siguiente documento presentara una forma adecuada y explicativa de cómo utilizar la aplicación.

En la cual se abarcarán los siguientes puntos: ¿Cómo interpretar la aplicación?,¿Cómo utilizar de manera apropiada?,¿Cuáles son los posibles errores que se pueden encontrar durante la ejecución del problema?,¿Qué funcionalidades satisface la aplicación?

Se mostrarán la forma de hacer el llamado a funciones, el cómo se le entregarán los parámetros y además la variedad de resultados que nos puede entregar.

Se planteará el procedimiento que se debe realizar llamar ciertas funciones que van a necesitar de la salida que entrega otra función.

CAPÍTULO 2. INSTRUCCIONES

Esta aplicación se lleva a cabo dentro de la aplicación *Dr racket*, que se puede conseguir descargándola desde el siguiente link:

- https://download.racket-lang.org/

Terminado el paso anterior, se procede abrir la aplicación, obteniendo lo que se muestra en la imagen:

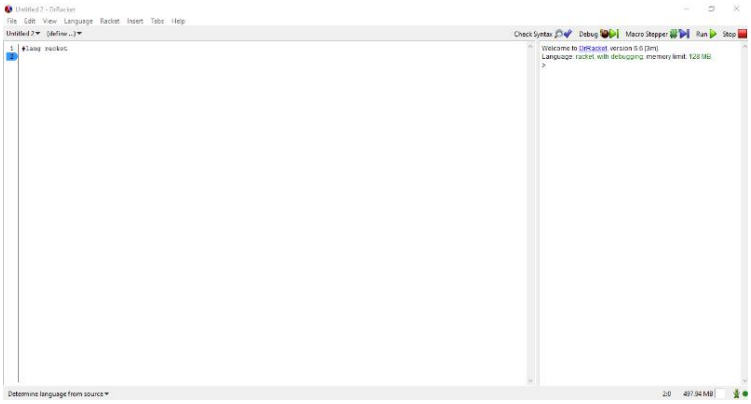

Figura CAPÍTULO 2. 1: Campo principal de *Dr.racket.*

Luego, se puede proceder a abrir alguna aplicación que desea interpretar, de la manera que se muestra en la siguiente imagen:

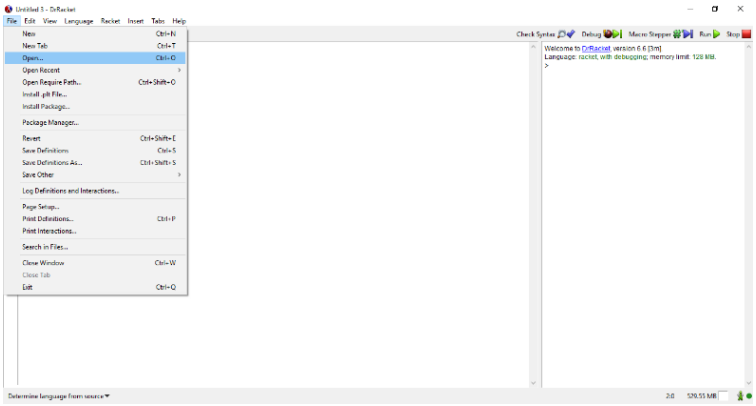

Figura CAPÍTULO 2. 2 : Representación de abrir un archivo.

De esta forma nos mostrara el código que tenga nuestro archivo que contenga la aplicación.

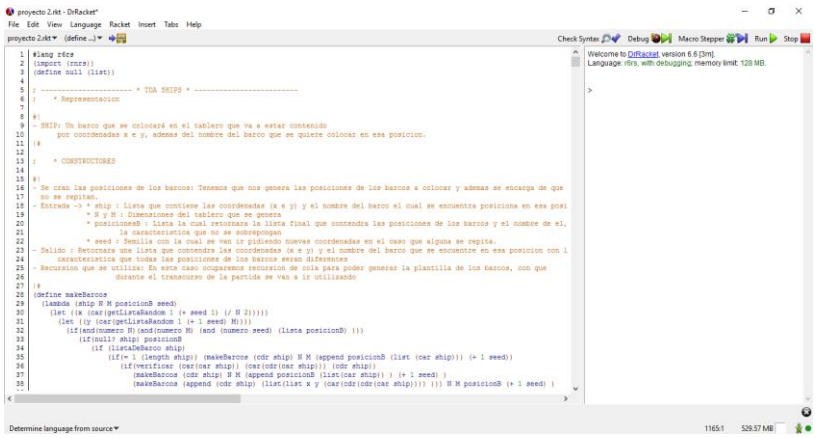

Figura CAPÍTULO 2. 3: Representación de código.

- De inmediato para poder interpretar el programa que se desea utilizar se tiene que utilizar el botón "*Run*", que se encuentra a la en la esquina superior izquierda, se puede apreciar en la siguiente imagen:

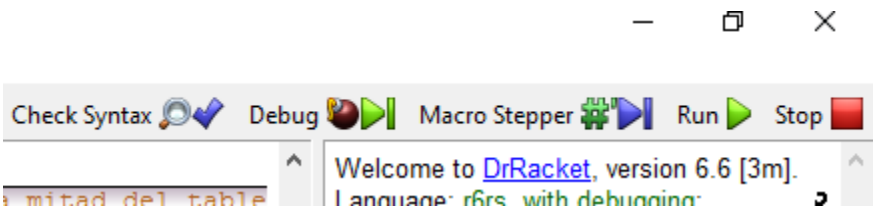

Figura CAPÍTULO 2. 4: Opciones del interprete *Dr.racket.*

  - Como se puede observar no solo la aplicación ofrece esa opción, sino que además se puede realizar lo siguiente:
    - *Debug*: Es una herramienta que se puede utilizar al momento de tener un error en el código, ya que sirve para ir recorriendo línea a línea el código e ir mostrando en los estados que se encuentra.
    - *Stop*: Con este botón podemos detener la interpretación de la aplicación en el momento que se desee.
    - *CkeckSyntax*: Sirve para verificar si la sintaxis del código está correcta.

CAPÍTULO 3. FUNCIONES

El primer paso importante, es ingresar el carácter con el cual será representado cada barco de la computadora.

```
#| 1. Defininmos los barcos |#

(define barcos (list #\c #\y #\k ))
```

Figura CAPÍTULO 3. 5: Definición de *Ship*.

Lo anterior nos da el primer paso para saber la cantidad de barcos que la computadora posicionará en N/2 de la matriz.

Como se tienen los barcos que se van a utilizar durante la partida con sus respectivas coordenadas podemos crear la matriz de N x M, esta se puede crear de dos formas distintas mediante: las cuales son recursión lineal o recursión de cola.

En la siguiente imagen se aprecia cómo se llama a la función *createBoardRL*:

```
#| 4. Creamos el tablero con las posiciones de los barcos de la computadora|#

(define board (createBoardRL 10 10 barco))
;(define board (createBoardRC 10 10 barco))
(display board)
```

Figura CAPÍTULO 3. 6: Llamado a función createBoardRL.

Se puede observar que recibe 3 parámetros (N, M, barco).

Luego tenemos que verificar si la matriz creada cumple con las condiciones elementales del juego, para poder verificar eso se llama a la función:

```
#| 5. luego verificamos ue el tablero este correctamente creado|#

(define check (checkBoard board))
(display check)
```

Figura CAPÍTULO 3. 7: Función *ckeckBoard*.

La cual recibe como parámetro la matriz que se le desea realizar la verificación.

Como se tiene creada la matriz de N x M y además es válida para poder comenzar una partida, podemos posiciones un barco del usuario en dicha matriz, realizando el llamado a la siguiente función:

```
;     * Funcion putShip

(define boardListo (putShip (putShip (putShip board '(6 6) #\h) '(9 7) #\t) '(6 9) #\o))
;(display boardListo)
```

Figura CAPÍTULO 3. 8: Llamado a la función *putShip*.

Recibe como parámetro (board, positions, barco)

Como se tiene creado la nueva matriz con los barcos del enemigo y de la computadora, podemos hacer que se realice un ataque dentro de la matriz, haciendo el llamado a la función que se muestra en la imagen:

```
;     * Funcion play

(define final (play boardListo #\h '(0 1) 156432))
(display final)

(display "\n")
```

Figura CAPÍTULO 3. 9: Llamado a la función *play*.

Recibe como parámetro nombre del *ship* que va a realizar el disparo, la posición donde se realizara, y una semilla (para poder generar las coordenadas random de disparo de la computadora).

Tener en cuenta que además de entregarnos el tablero con los ataques realizados nos entregara el historial de la última jugada realizada en el juego

Se puede realizar el llamado a una función que muestra la matriz creada de N x M, en una forma que se puede apreciar de mejor manera.

Haciendo el llamado a la función que se muestra en la siguiente imagen:

```
;     * Funcion board->string

(define boardString (board->string (car final) 0))
(display boardString)
```

Figura CAPÍTULO 3. 10: Llamada a la función *Board->string*.

Recibe como parámetro *board.*

También se puede realizar el llamado a funciones que entregan la matriz creada en formato *XML* y *JSON*.

Estos llamados se realizan como muestra la siguiente imagen:

```
;     * Funcion que muestran en la matriz en otros formatos

(define Board->json (board->xml board ))
(define Board->xml (board->xml (car final) ))

(display Board->json)
(display "\n")
(display Board->xml)
```

Figura CAPÍTULO 3. 11: Llamado a funciones que entregan la matriz en distintos formatos.

CAPÍTULO 4. POSIBLES ERRORES

La aplicación fue creada para que las funciones revisan ciertos parámetros en específicos, por lo que si no son los esperados existe una alta probabilidad que la aplicación entrega un resultado booleano denegando lo solicitado o simplemente falle.

A continuación, se descompondrán algunos de los posibles errores que se pueden encontrar en la aplicación:

- La aplicación fue programada para que las funciones revisan los parámetros correctos, en caso contrario retorna un valor booleano #f, que significa que alguno de los parámetros ingresados no pertenece a lo solicitado.
- Si la función está esperando recibir como parámetro un numero entero y se le ingresa una letra o palabra, el programa mostrará un error que produce por problemas de datos incompatibles.