

UNIVERSIDAD DE SANTIAGO DE CHILE  
FACULTAD DE INGENIERÍA  
Departamento de Ingeniería Informática



## **CONECTA 4**

PROYECTO DE LABORATORIO 1 – PARADIGMA FUNCIONAL

Paradigmas de Programación

Matias Fernando Ramírez Escobar 21.484.373-0

matias.ramirez.e@usach.cl

**Sección: 13204-0 A-1**

**Fecha de entrega: 1/11/24**

**Carrera: Ingeniería de ejecución  
en computación e informática**

# Tabla de contenidos

Introducción.....	2
El problema.....	3
Scheme y su paradigma .....	3
Análisis del problema.....	3
Diseño de la solución.....	4
Aspectos de implementación .....	5
Instrucciones de uso.....	6
Resultados y autoevaluación .....	6
Conclusiones del trabajo.....	7
Referencias .....	7
Anexos.....	8

## Introducción

En el dinámico campo de la programación, algunos lenguajes se han consolidado como referentes en la industria. No obstante, la versatilidad de un ingeniero informático se define por su capacidad para dominar diversos paradigmas de programación. Este proyecto tiene como objetivo explorar el paradigma funcional a través del lenguaje Scheme, mediante el desarrollo de un juego de estrategia educativo. La elección de Scheme permite aprovechar su elegancia y expresividad

para diseñar un juego sólido y eficiente, que fomente el desarrollo de habilidades cognitivas, sensoriales, sociales y emocionales en los niños, al tiempo que promueve el razonamiento lógico, la imaginación y la creatividad.

## **El problema**

El desafío central de este proyecto es desarrollar una versión del clásico juego Conecta 4 utilizando el lenguaje de programación Scheme. La dificultad principal radica en representar adecuadamente el tablero e implementar las mecánicas del juego, respetando las reglas del juego original, todo dentro del paradigma funcional. El uso de Scheme presenta retos, ya que no es un lenguaje tradicionalmente asociado con el desarrollo de juegos, pero ofrece la oportunidad de explorar su expresividad y potencial dentro de este ámbito.

El objetivo de este proyecto es implementar Conecta 4 en Scheme, utilizando las herramientas y estrategias del paradigma funcional. A través de este enfoque, se busca demostrar el dominio de Scheme y su aplicabilidad en el desarrollo de un juego que sea lo más similar posible al original, mientras se aprovechan las ventajas que ofrece la programación funcional, como la claridad en la modelación de las reglas y la dinámica del juego.

## **Scheme y su paradigma**

Scheme, un lenguaje derivado de Lisp creado en el MIT en la década de 1970, se centra en la programación funcional. En este paradigma, las funciones son la principal herramienta para manipular datos, mientras que las listas constituyen la estructura de datos fundamental. A diferencia de otros lenguajes que se basan en la asignación y los bucles, Scheme utiliza la recursión como técnica principal para construir programas concisos y eficientes.

Su sintaxis es elegante y simple, basada en el uso de paréntesis y la notación prefija, lo que facilita tanto la lectura como la escritura de código. Además, Scheme trata a las funciones como valores de primera clase, lo que significa que pueden ser manipuladas como cualquier otro dato. La ausencia de un estado mutable lo convierte en una herramienta ideal para aprender los principios de la programación funcional.

Estas características hacen de Scheme un lenguaje adecuado para implementar el juego de estrategia Conecta 4, aprovechando sus primitivas y su eficiente representación de estructuras mediante listas.

## **Análisis del problema**

Para resolver el problema planteado, es necesario implementar el juego Conecta 4 utilizando Scheme. Esto requiere una abstracción específica de cada uno de los

elementos y funcionalidades del juego, adecuándolos a las capacidades y limitaciones del lenguaje. El uso eficiente de las funciones, con una clara comprensión de su dominio (los parámetros que reciben) y su recorrido (las estructuras o tipos de datos que devuelven), será fundamental para lograr una implementación exitosa.

Este proyecto debe cumplir con los siguientes aspectos clave para simular correctamente una partida de Conecta 4:

1. Creación y gestión de partidas:
  - Iniciar una nueva partida.
  - Guardar el estado de una partida actual.
  - Configurar el modo de juego (jugador 1 vs jugador 2 o jugador vs IA).
2. Gestión de Jugadores:
  - Registrar nuevos jugadores con su nombre y color de fichas.
  - Modificar la información de los jugadores que estén en la partida.
  - Consultar estadísticas de los jugadores (victorias, derrotas, empates).
3. Desarrollo del Juego:
  - Realizar movimientos colocando fichas en el tablero.
  - Validar los movimientos de acuerdo con las reglas del juego.
  - Detectar el resultado de la partida: victoria, empate o continuación.
  - Mostrar el estado actual del tablero.
4. Integración con IA generativa:
  - Implementar un modo de juego usuario vs IA, donde la IA responda a través de una consulta a la API gratuita de Google AI Studio (Google Gemini).
  - Resumir las estadísticas del juego en lenguaje natural, proporcionando un análisis de las victorias y derrotas de cada jugador.
5. Análisis del Juego:
  - Generar un historial de movimientos realizados durante la partida.

Estos apartados permiten cubrir todos los requisitos necesarios para simular una partida funcional de Conecta 4. Los tres primeros son esenciales para representar una partida común, mientras que las funcionalidades adicionales, como el historial de jugadas, ofrecen una ventaja que no está presente en el juego físico. Este historial permite analizar jugadas anteriores y diseñar estrategias para futuros enfrentamientos, enriqueciendo la experiencia del usuario.

## **Diseño de la solución**

Para representar el juego Conecta 4 de manera efectiva, es esencial identificar los principales Tipos Abstractos de Datos (TDA) que serán implementados mediante diversas funciones, ya sean representativas, constructoras, de pertenencia,

selectoras, modificadoras, u otras. Con este enfoque, el proyecto se divide en cuatro TDA principales (ver anexo 1): el TDA board (tablero), el TDA piece (ficha), el TDA player (jugador) y el TDA game (juego). El TDA game será el encargado de manipular y coordinar los datos de los otros TDA para simular una partida completa.

Un aspecto importante del diseño es que, al crear un nuevo juego, se requiere un tablero vacío y la definición de los jugadores. Durante cada turno, el jugador debe especificar tanto el jugador activo como la columna en la que desea colocar su ficha, la cual siempre caerá en la última casilla disponible, de arriba hacia abajo. El juego verificará internamente que sea el turno correcto del jugador, y actualizará el estado del juego tras cada movimiento. Estas actualizaciones incluyen:

- Colocar la ficha en el tablero.
- Cambiar el turno.
- Disminuir el número de fichas restantes para el jugador que realizó el movimiento.
- Actualizar el historial de movimientos.
- Verificar si hay un ganador o si el juego ha terminado en empate.
- Si el juego ha concluido, actualizar las estadísticas de ambos jugadores según corresponda.

Para lograr una implementación clara y eficiente, se emplearon diversas estrategias, como la creación de funciones auxiliares dentro de las funciones principales, con el fin de mantener un código limpio y fácil de comprender. Además, se utilizó la resolución declarativa en las funciones, aplicando recursión natural o de cola según fuera necesario para satisfacer los requisitos del proyecto. Esto se realizó respetando el dominio, el recorrido y el tipo de recursión a emplear, siempre siguiendo las bases del paradigma funcional.

## **Aspectos de implementación**

Para la implementación de este proyecto, se utilizó el intérprete DrRacket, una excelente opción para trabajar con Scheme debido a su interfaz clara y su funcionalidad de "debug", que permite realizar un seguimiento detallado de los pasos del programa. Esta capacidad es especialmente valiosa en un proyecto extenso que involucra múltiples Tipos Abstractos de Datos (TDA) y sus respectivas funciones, ayudando a identificar y corregir errores durante el desarrollo.

El enfoque de implementación se basa en el encapsulamiento de los distintos TDAs. Todos ellos están contenidos dentro del TDA game, mientras que los TDAs internos (board, piece, y player) tienen sus propias funciones. Este diseño modular permite centrarse en los resultados que las funciones deben producir (recorrido), en lugar de preocuparse por los detalles internos de su funcionamiento, lo que resulta en un código más legible y fácil de manipular.

El proyecto fue desarrollado íntegramente en el lenguaje **Racket**, un derivado de Scheme, utilizando la versión **DrRacket 8.10**. Este software actúa como compilador y es capaz de ejecutar las funciones dentro de su propio entorno interactivo (IDLE), lo que facilita la visualización de los resultados y la depuración del código.

Se utilizó la documentación oficial de DrRacket como medio de aprendizaje para comprender correctamente el lenguaje y sus herramientas. Además, al inicio del proyecto, se emplearon herramientas de IA para interpretar errores de sintaxis comunes y mensajes de error específicos en la consola. Sin embargo, nunca se utilizó ningún código generado por estas herramientas.

## Instrucciones de uso

Este proyecto ha sido diseñado para ser fácil de utilizar. Una vez abierto el script de pruebas y presionando el botón "Run", el programa ejecutará automáticamente una partida de Conecta 4. A continuación, se detallan los pasos para ejecutar correctamente el programa:

1. Instalar **DrRacket** en el equipo.
2. Abrir la carpeta "**Lab1\_214843730\_MatiasRamirezEscobar**" y verificar la existencia de los 7 archivos **.rkt** necesarios (ver Anexo 2).
3. Abrir el archivo "**main\_212788287\_EspinozaBarria.rkt**" en DrRacket.
4. Hacer clic en el botón "**Run**" dentro del entorno de desarrollo.
5. Ejecutar la función de verificación deseada en el IDLE (ver Anexo 3).

Los ejemplos incluidos en los scripts de prueba permiten simular diferentes tipos de victorias, como la victoria diagonal, vertical y horizontal. Además, es posible iniciar una nueva partida desde cualquiera de estos scripts. Sin embargo, es importante seguir la estructura de ejecución correcta (ver Anexo 4) para evitar errores.

Se espera que el programa funcione correctamente en todos los Requisitos Funcionales (RF) implementados, siempre y cuando el usuario no introduzca datos inválidos. De lo contrario, podrían generarse errores. Algunos ejemplos de situaciones que pueden causar errores incluyen:

- Ingresar incorrectamente el color de la ficha escogida.
- Proporcionar los parámetros en un orden incorrecto al crear un jugador o al llamar a las funciones "game" o "game-player-set-move".
- Asignar el mismo color a ambos jugadores.
- Ingresar una columna no válida (un número que no esté entre 1 y 7).

## Resultados y autoevaluación

Los resultados correspondientes a los RF y RNF se detallan en los anexos 5 en adelante. Sin embargo, de manera general, se puede afirmar que el programa funciona correctamente hasta el RF 18, siempre y cuando se respeten el orden de

los parámetros y se ingresen los datos válidos tal como se mencionó en la sección anterior.

Bajo estas condiciones, el proyecto ha demostrado ser exitoso en la implementación del juego Conecta 4 utilizando el paradigma funcional. El código cumple con los objetivos planteados, logrando una simulación completa del juego sin presentar errores significativos. Esto confirma que las técnicas utilizadas en el diseño y desarrollo del proyecto, como el manejo de TDAs y la recursión, fueron adecuadas para modelar de manera efectiva la lógica del juego en Scheme.

En cuanto a la autoevaluación, se considera que se ha logrado cumplir con todos los RF esenciales, garantizando un desarrollo estable y eficiente. Los elementos que han sido implementados cumplen con las expectativas iniciales del proyecto, ofreciendo una experiencia de juego fluida y sin errores importantes, siempre que se sigan las instrucciones de uso correctamente.

## Conclusiones del trabajo

Este trabajo representó un desafío significativo, especialmente al pasar de experiencias previas centradas en el paradigma procedural imperativo a la programación funcional. Este cambio de enfoque requirió una transformación en la forma de pensar sobre estructuras de datos y su manipulación. Al principio, se percibió como una limitante, ya que la abstracción del paradigma funcional puede resultar desconcertante. Sin embargo, a medida que se profundizó en el tema, se empezaron a descubrir nuevas posibilidades. Lo que inicialmente parecía complicado se convirtió en una oportunidad para comprender mejor el potencial de la programación funcional, que, a su vez, demostró ser intuitiva y fácil de entender, especialmente en relación con las distintas formas de recursión.

El proyecto se completó en un 90%, ya que la implementación de la IA de Google Gemini no se llevó a cabo debido a limitaciones de tiempo. Tanto la investigación sobre su uso como la priorización del modo de juego entre jugadores fueron obstáculos en este aspecto. A pesar de la falta de estas dos funcionalidades, se siente una gran satisfacción por haber alcanzado el objetivo principal del proyecto. No obstante, es crucial continuar investigando y formándose en esta tecnología de inteligencia artificial, que es cada vez más demandada en el campo de la programación.

## Referencias

1. Racket. (s. f.). <https://racket-lang.org/>
2. The racket reference. (s. f.). <https://docs.racket-lang.org/reference/>
3. GitHub · Build and ship software on a single, collaborative platform. (2024). GitHub. <https://github.com/>

4. C, D. E. (2023, 15 agosto). Fundamentos de la programación funcional - Diego Esteban C - Medium. Medium.  
<https://medium.com/@diego.coder/fundamentos-de-la-programaci%C3%B3n-funcional-137366c85279>
5. Kuhn, T. S. (1962). La estructura de las revoluciones científicas

## Anexos

1.-













Nombre TDA	Dato Abstracto	Estructura	Funciones asociadas al TDA
Board	Representación de un tablero donde se colocarán las fichas y se usara para verificar ganadores u empate.	Lista de Listas vacías.	board, board-can-play?, board-set-play-piece, board-check-vertical-win, board-check-horizontal-win, board-check-diagonal-win, board-who-is-winner
Player	Representación de un jugador con sus propios atributos, además de poder actualizar estadísticas en caso de victoria derrota o empate.	Lista que contiene el id, nombre, color, victorias, derrotas, empates.	player, player-update-stats
Piece	Representación de una pieza para jugar sobre el tablero que va asociada a cada jugador según su color.	Lista con carácter representativo para el color escogido.	piece



Game	Representación del juego conecta 4, la cual se forma a partir del conjunto de este TDA y los anteriores descritos, el cual puede colocar una ficha y hace las respectivas verificaciones necesarias para el correcto funcionamiento del programa.	Lista contenedora de dos jugadores y un tablero en el cual se juega la partida.	game, game-is-draw?, game-get-current-player, game-get-board, game-set-end, game-player-set-move, game-history
------	---	---	--

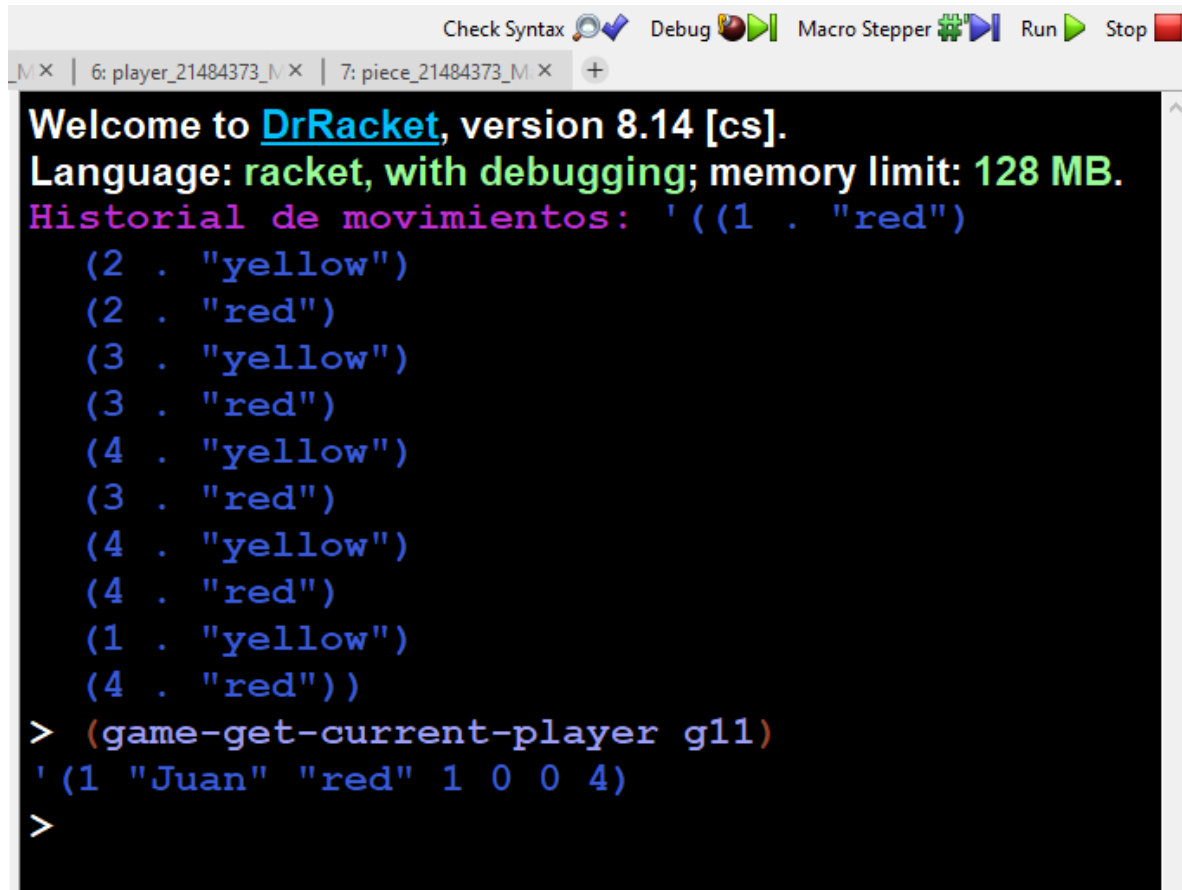
Tabla de TDAs implementados.

2.-

Nombre	Fecha de modificación	Tipo	Tamaño
 .git	29-10-2024 22:29	Carpeta de archivos	
 Autoevaluacion_21484373_MatiasRamire...	31-10-2024 14:56	Documento de te...	2 KB
 board_21484373_MatiasRamirezEscobar.rkt	31-10-2024 18:05	Racket Document	8 KB
 Comandos_21484373_MatiasRamirezEsco...	31-10-2024 15:11	Documento de te...	2 KB
 game_21484373_MatiasRamirezEscobar.rkt	31-10-2024 18:05	Racket Document	7 KB
 Instrucciones_21484373_MatiasRamirezEs...	31-10-2024 17:22	Documento de te...	4 KB
 piece_21484373_MatiasRamirezEscobar.rkt	31-10-2024 18:05	Racket Document	1 KB
 player_21484373_MatiasRamirezEscobar.rkt	31-10-2024 18:05	Racket Document	2 KB
 README_21484373_MatiasRamirezEscob...	31-10-2024 17:29	Documento de te...	1 KB
 script_base_21484373_MatiasRamirezEsco...	31-10-2024 18:05	Racket Document	2 KB
 script1_21484373_MatiasRamirezEscobar....	31-10-2024 18:09	Racket Document	2 KB
 script2_21484373_MatiasRamirezEscobar....	31-10-2024 18:37	Racket Document	2 KB

Carpeta que contiene los archivos .rkt.

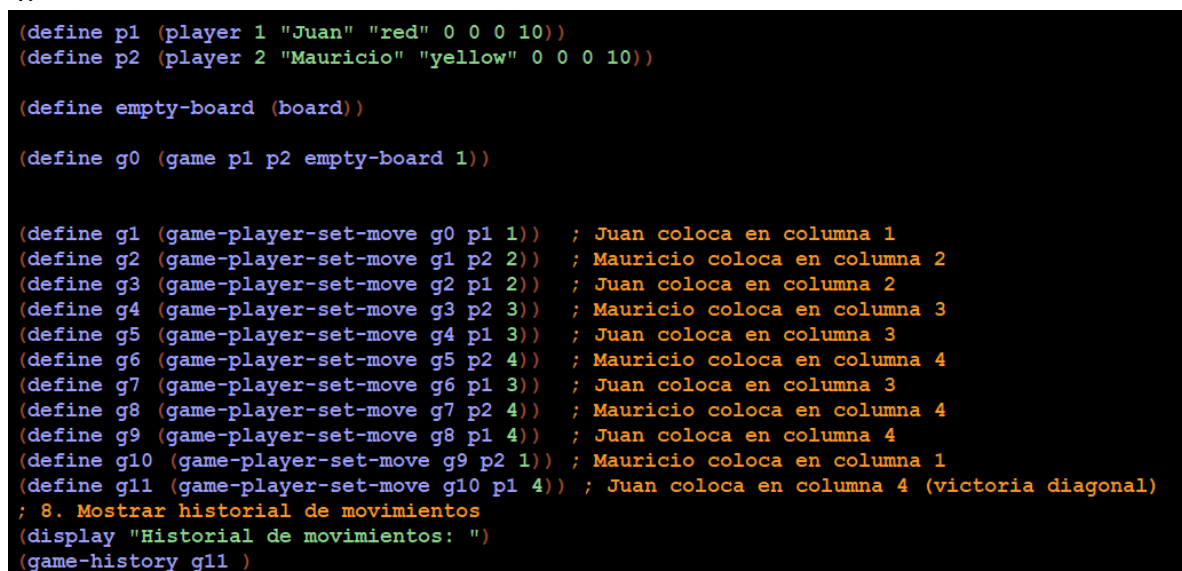
3.-



```
Check Syntax  Debug  Macro Stepper  Run  Stop
_M x | 6: player_21484373_M x | 7: piece_21484373_M x | +
Welcome to DrRacket, version 8.14 [cs].
Language: racket, with debugging; memory limit: 128 MB.
Historial de movimientos: '((1 . "red")
  (2 . "yellow")
  (2 . "red")
  (3 . "yellow")
  (3 . "red")
  (4 . "yellow")
  (3 . "red")
  (4 . "yellow")
  (4 . "red")
  (1 . "yellow")
  (4 . "red"))
> (game-get-current-player g11)
'(1 "Juan" "red" 1 0 0 4)
>
```

Ejemplo de función de verificación.

4.-



```
(define p1 (player 1 "Juan" "red" 0 0 0 10))
(define p2 (player 2 "Mauricio" "yellow" 0 0 0 10))

(define empty-board (board))

(define g0 (game p1 p2 empty-board 1))

(define g1 (game-player-set-move g0 p1 1)) ; Juan coloca en columna 1
(define g2 (game-player-set-move g1 p2 2)) ; Mauricio coloca en columna 2
(define g3 (game-player-set-move g2 p1 2)) ; Juan coloca en columna 2
(define g4 (game-player-set-move g3 p2 3)) ; Mauricio coloca en columna 3
(define g5 (game-player-set-move g4 p1 3)) ; Juan coloca en columna 3
(define g6 (game-player-set-move g5 p2 4)) ; Mauricio coloca en columna 4
(define g7 (game-player-set-move g6 p1 3)) ; Juan coloca en columna 3
(define g8 (game-player-set-move g7 p2 4)) ; Mauricio coloca en columna 4
(define g9 (game-player-set-move g8 p1 4)) ; Juan coloca en columna 4
(define g10 (game-player-set-move g9 p2 1)) ; Mauricio coloca en columna 1
(define g11 (game-player-set-move g10 p1 4)) ; Juan coloca en columna 4 (victoria diagonal)
; 8. Mostrar historial de movimientos
(display "Historial de movimientos: ")
(game-history g11 )
```

Ejemplo de correcta ejecución del programa (script base).

5.-

Requerimientos	Grado de implementación	Pruebas realizadas	Pruebas exitosas %	Pruebas fracasadas %	Anotación
TDA's	1	4 TDA's implementados	100	0	
TDA Player - constructor	1	5 creaciones de player	100	0	
TDA Piece - constructor	1	20 creaciones de pieza	99	1	El fracaso fue cuando se intentó usar un color no permitido.
TDA Board - constructor	1	30 creaciones de tablero	100	0	
TDA Board - otros - sePuedeJugar?	1	10 consultas	100	0	
TDA Board - modificador - jugar ficha	1	40 fichas colocadas	95	5	Los intentos fallidos son de momentos en los que se refinaba el funcionamiento de la función.
TDA Board - otros - verificar victoria vertical	1	15 consultas	100	0	
TDA Board - otros - verificar victoria horizontal	1	15 consultas	100	0	
TDA Board - otros - verificar victoria diagonal	1	15 consultas	100	0	
TDA Board - otros - entregarGanador	1	20 consultas	100	0	
TDA Game - constructor	1	15 games contruidos	100	0	
TDA Game - otros - history	1	15 llamadas	100	0	
TDA Game - otros - esEmpate?	1	15 verificaciones	100	0	
TDA Player - otros - actualizarEstadisticas	1	10 actualizaciones	100	0	

TDA Game - selector - getCurrentPlayer	1	10 llamados a el getter	100	0	
TDA Game - selector - board-get-state	1	15 llamadas al getter	100	0	
TDA Game - modificador - game-set-end	1	10 términos de un juego	100	0	
TDA Game - modificador - realizarMovimiento	1	40 veces que se juega	100	0	
TDA Game - modificador - game-ai-set-move	1	0	0	0	No se logró implementar por límites de tiempo y falta de atención a la función.
TDA Game - otros - game-ai-explain-history	1	0	0	0	No se logró implementar por límites de tiempo y falta de atención a la función.

#### Autoevaluación de requerimientos funcionales

6.-

RNF	Implementacion	Anotaciones
Autoevaluación	1	Presente en la carpeta del proyecto.
Lenguaje	1	Lenguaje requerido Scheme/Racket.
Versión	1	6.1 o superior.
Standard	1	Librería estándar.
No variables	1	No se emularon variables.
Documentación	1	Toda función presenta su dominio, recorrido, función, tipo de recursión y su tipo.
Dom->Rec	1	Presente en todas las funciones.
Organización	1	Cada TDA tiene su archivo correspondiente.
Historial	1	Se utilizo GitHub respetando los 10 commits mínimos en dos semanas.

#### Autoevaluación de los requerimientos no funcionales