

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



PROYECTO DE LABORATORIO 1 – PARADIGMA FUNCIONAL

Paradigmas de Programación

Gerardo Ignacio Pérez Encina 21.343.296-6

Sección:

13204-0-A-1

Profesor:

Edmundo Leiva Lobos

Fecha:

31 de octubre del 2024

TABLA DE CONTENIDO

| | |
|---|---|
| TABLA DE CONTENIDO | 2 |
| 1. INTRODUCCIÓN..... | 3 |
| 2. DESCRIPCIÓN DE LA PROBLEMÁTICA..... | 3 |
| 2.1. PROBLEMA..... | 3 |
| 2.2. DESCRIPCIÓN DEL PARADGIMA..... | 4 |
| 3. ANÁLISIS DEL PROBLEMA | 4 |
| 4. DISEÑO DE LA SOLUCIÓN | 5 |
| 5. CONSIDERACIONES DE IMPLEMENTACIÓN..... | 6 |
| 5.1. ESTRUCTURA DEL PROYECTO | 6 |
| 5.2. BIBLIOTECAS EMPLEADAS..... | 6 |
| 5.3. INTERPRETE USADO | 6 |
| 6. INSTRUCCIONES DE USO..... | 6 |
| 7. RESULTADOS Y AUTOEVALUACIÓN..... | 6 |
| 8. CONCLUSIONES..... | 7 |
| 9. REFERENCIAS..... | 7 |
| 10. ANEXO..... | 8 |

1. INTRODUCCIÓN

Conocer diferentes paradigmas de programación es esencial para que los programadores desarrollen soluciones efectivas ante problemáticas considerando todos los puntos de vista posibles y eligiendo los mejores métodos para escribir sus programas.

En el presente laboratorio se nos presenta la creación del famoso juego Conecta 4, mediante un programa en el lenguaje Racket/Scheme bajo el paradigma funcional de programación.

El documento consta de introducción, descripción del problema, análisis del problema, diseño de la solución, consideraciones de implementación, instrucciones de uso, resultados y autoevaluación, y conclusiones.

2. DESCRIPCIÓN DE LA PROBLEMÁTICA

2.1. PROBLEMA

Cuando investigamos más sobre el juego de Conecta 4, nos damos cuentas que nuestro problema es la implementación de este, pero primero hablemos sobre que es Conecta 4, este juego trata sobre un tablero de 6x7 el cual se juega con dos jugadores, cada jugador tiene una cantidad de piezas y en cada turno solo pueden colocar una pieza, la cual cae desde arriba hasta el fondo o hasta donde sea posible, para poder determinar un ganador se debe de hacer una línea vertical, horizontal o diagonal.

2.2. DESCRIPCIÓN DEL PARADGIMA

Para entender los métodos a emplear en la resolución del problema, es útil conocer algunos conceptos claves que facilitan su comprensión. A continuación, se presentan y describen:

1.- **Paradigma funcional:** Es un enfoque de programación en el cual los problemas se resuelven a través de funciones y su composición, sin usar asignación de variables. Este es un paradigma declarativo, lo que significa que se especifica qué debe hacer el programa en lugar de como hacerlo.

2.- **Recursión:** Es una técnica en la cual una función se llama a sí misma directa o indirectamente para resolver un problema dividiéndolo en subproblemas más pequeños. Es especialmente útil en programación funcional para iterar sobre estructuras de datos o realizar cálculos complejos sin utilizar bucles.

3.- **Funciones de orden superior:** Son funciones que pueden recibir otras funciones como parámetros y/o devolver funciones como resultado. Este concepto permite una mayor abstracción y flexibilidad al diseñar soluciones, ya que se pueden construir funciones genéricas y reutilizables.

4.- **Funciones lambda:** También conocidas como funciones anónimas, son funciones definidas de manera breve que no están asociadas a un nombre específico. En Scheme, se usan comúnmente para escribir expresiones simples y rápidas en lugar de definir funciones formales.

5.- **Currificación:** Es una técnica en programación funcional que transforma una función que toma múltiples argumentos en una secuencia de funciones, cada una de las cuales toma un solo argumento. Esto permite la creación de funciones parcialmente aplicadas, que son útiles para componer y reutilizar código.

3. ANÁLISIS DEL PROBLEMA

Para solucionar el problema, se desea crear el juego Conecta 4 en el lenguaje de programación Scheme, bajo el paradigma funcional declarativo. Esto quiere decir que se debe crear un programa que, mediante funciones, se consiga poder jugar una partida de Conecta 4 sin problemas.

Por consiguiente, se deben considerar diferentes TDAs que logran obtener los elementos importantes del juego para poderlos trabajar dentro del programa. Donde se declaren todas las funciones que puedan hacer que el Conecta 4 funcione.

El Conecta 4 está constituido por: Jugadores, tablero y piezas. Y como requisitos, el programa de debe cumplir con las siguientes funcionalidades: Creación y Gestión de la partida (Crear partida, guardar partida, jugador contra jugador, jugador contra IA), gestión de jugadores (Registrar nuevo jugador, modificar información de algún jugador existente, consultar estadísticas de los jugadores), desarrollo del juego (Realizar los movimientos, validar los movimientos según las reglas del juego, detectar victoria, empate o continuación del juego, mostrar el estado actual del tablero).

4. DISEÑO DE LA SOLUCIÓN

Para implementar el juego Conecta 4, surge la necesidad de crear diferentes abstracciones de elementos mediante TDAs. De modo de ir conteniendo tipos de datos abstractos dentro de un gran TDA que represente el juego

Debido a que el lenguaje Scheme está relacionado con las listas de elementos, los TDAs creados utilizan listas como estructura de datos. A continuación, se mencionan los tipos de datos abstractos usados, y en el Anexo 1 se expone la explicación sobre cómo están constituidos: Player, Board, Piece, Game.

Algunos elementos importantes que destacar son las reglas necesarias para validar una victoria en el juego. Para que un jugador gane, es imprescindible lograr una alineación de cuatro piezas consecutivas del mismo color en cualquiera de las siguientes direcciones: horizontal, vertical o diagonal. Además, el tablero está compuesto por una estructura de columnas, y cada jugador debe colocar su pieza desde la parte superior de una columna, dejándola caer hasta una posición que no esté ocupada por la pieza, lo que implica que las piezas se apilan una sobre otra. Por otro lado, si el tablero se llena completamente sin que ningún jugador logre una alineación válida, el juego termina en empate. Es importante considerar también que, en cada turno, un jugador puede colocar solo una pieza, y la alternancia de turnos es estricta para garantizar la fluidez del juego. Estas reglas permiten estructurar la lógica del juego y definir los estados válidos en cada etapa de la partida.

Como elementos de programación se utilizó la recursión tanto natural como de cola para trabajar con las listas y así también poder cumplir con los requerimientos funcionales que pedían emplear algún tipo de recursión en particular, o por el contrario resolver procedimientos de forma declarativa; por lo que para esto se usó funciones como map o list-ref de la librería estándar. Procurando siempre respetar los principios de la programación funcional.

5. CONSIDERACIONES DE IMPLEMENTACIÓN

5.1. ESTRUCTURA DEL PROYECTO

El proyecto se basa en la encapsulación de diferentes TDAs dado que todos están contenidos bajo el TDA principal llamado “game”. Donde cada TDA tienen sus propias funciones que facilitan la operación de estos.

5.2. BIBLIOTECAS EMPLEADAS

El lenguaje Racket/Scheme posee varias librerías con funciones disponibles para el usuario, sin embargo, fueron utilizadas sólo las primitivas del dialecto; en particular la librería estándar del lenguaje Scheme (R6RS).

5.3. INTERPRETE USADO

El programa fue totalmente desarrollado en el lenguaje Racket (derivado de Scheme), escrito en el software DrRacket 8.14 como IDE; el que también puede compilar el programa y ejecutar las funciones en su propia consola de comando.

6. INSTRUCCIONES DE USO

Para poder poner a prueba el programa, debe de seguir los siguientes pasos:

- 1.- Abrir la carpeta “Laboratorio1_21343296_PerezEncina” y verificar la existencia de los archivos (ver Anexo 2)
- 2.- Abrir el archivo “script_base_21343296_PerezEncina” en el IDE DrRacket (ver Anexo 3)
- 3.- Presionar el botón “Run” en la esquina superior derecha del IDE (ver Anexo 4)
- 4.- Posterior a eso verá como el programa empieza a funcionar. (ver Anexo 5)

Para poder visualizar los otros scripts deberá de repetir el paso 2 y 3, pero con la excepción de que deberá de elegir los otros dos archivos (script2_21343296_PerezEncina y script3_21343296_PerezEncina)

7. RESULTADOS Y AUTOEVALUACIÓN

Los resultados correspondientes a las RF se detallan en el archivo

“autoevaluación_21343296_PerezEncina”. Sin embargo, se puede afirmar que el programa funciona correctamente hasta la RF 18, siempre y cuando se respeten las instrucciones de uso.

Habiendo dicho eso, el proyecto ha demostrado ser exitoso en la implementación del juego

Conecta 4. El código cumple con los objetivos planteados, logrando así una simulación completa del juego.

8. CONCLUSIONES

El uso del paradigma funcional para abordar el problema planteado demostró ser una herramienta efectiva en términos de claridad y modularidad. Las funciones de orden superior y la recursión permitieron descomponer el problema en subproblemas más simples, fomentando un diseño lógico y estructurado. Asimismo, la composición de funciones facilitó la reutilización de código y la creación de soluciones generales aplicables a diferentes casos.

Sin embargo, el paradigma funcional también presentó limitaciones y dificultades. Una de las principales fue la necesidad de comprender profundamente conceptos como la currificación y las funciones lambda, los cuales pueden no resultar intuitivos para quienes están acostumbrados a paradigmas imperativos. Además, la ausencia de estados mutables complicó ciertas implementaciones, particularmente en problemas que requerían un seguimiento explícito de cambios en estructuras complejas.

En cuanto a los alcances, el paradigma funcional probó ser ideal para resolver problemas relacionados con estructuras de datos inmutables y cálculos matemáticos. No obstante, en problemas donde se requiere un manejo extensivo de estados o interacciones dinámicas, su uso puede volverse menos eficiente o más complejo de implementar.

9. REFERENCIAS

- Kuhn, T. S. (1962). *La estructura de las revoluciones científicas*.
- Spigariol, L. (2005). *Fundamentos teóricos de los Paradigmas de Programación*. Buenos Aires: Facultad Regional Buenos Aires Universidad Tecnológica

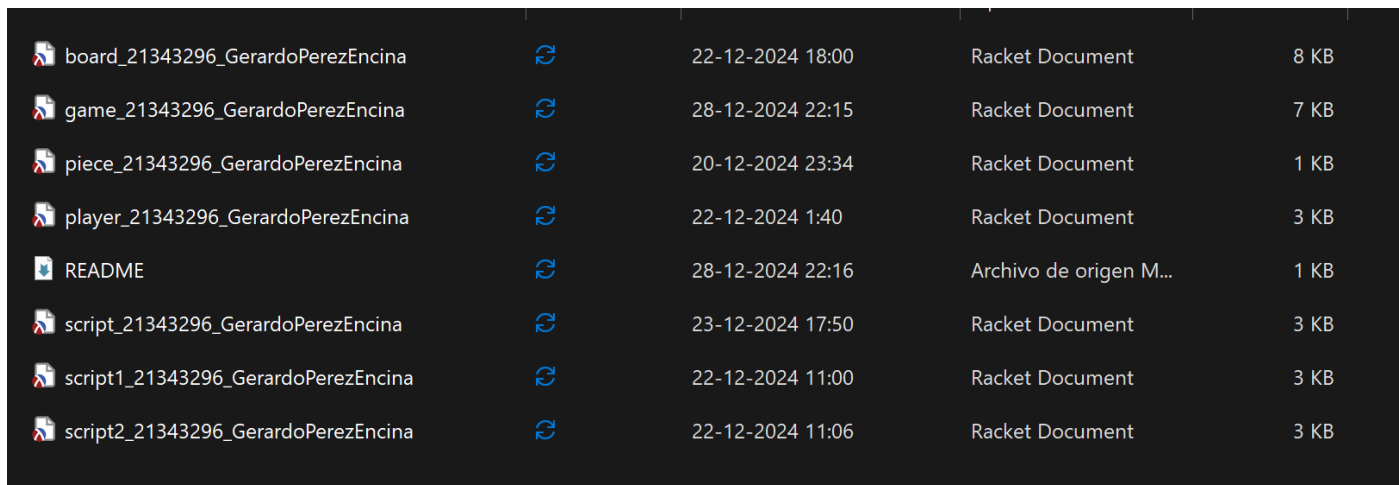
10. ANEXO

1.-

| Nombre TDA | Dato abstracto | Estructura | Operaciones relevantes |
|------------|---|---|---|
| Board | Representación del tablero, donde se colocarán las fichas | Filas, Columnas, Lista de listas (tablero) | Crear 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 del jugador junto a sus atributos | Lista(Id, name, color, wins, losses, draws, remaining-pieces) | Crear Player, player-update-stats |
| Piece | Representación de la pieza para jugar sobre el tablero | Lista(Color) | Crear Piece |
| Game | Representación del juego Conecta 4, formándose a partir de las TDAs mencionadas anteriormente | Lista(player1, player2, board, current-turn, historial) | Crear Game, game-history, game-is-draw?, game-get-current-player, game-get-board, game-set-end, game-player-set-move |

Tabla 1: Especificación de TDAs usados

2.-



















| | | | | |
|---|---|------------------|------------------------|------|
|  board_21343296_GerardoPerezEncina |  | 22-12-2024 18:00 | Racket Document | 8 KB |
|  game_21343296_GerardoPerezEncina |  | 28-12-2024 22:15 | Racket Document | 7 KB |
|  piece_21343296_GerardoPerezEncina |  | 20-12-2024 23:34 | Racket Document | 1 KB |
|  player_21343296_GerardoPerezEncina |  | 22-12-2024 1:40 | Racket Document | 3 KB |
|  README |  | 28-12-2024 22:16 | Archivo de origen M... | 1 KB |
|  script_21343296_GerardoPerezEncina |  | 23-12-2024 17:50 | Racket Document | 3 KB |
|  script1_21343296_GerardoPerezEncina |  | 22-12-2024 11:00 | Racket Document | 3 KB |
|  script2_21343296_GerardoPerezEncina |  | 22-12-2024 11:06 | Racket Document | 3 KB |

Ilustración 1: Carpeta contenedora de archivos

3.-

```
2 |lang racket
3
4 (require "player_21343296_GerardoPerezEncina.rkt")
5 (require "piece_21343296_GerardoPerezEncina.rkt")
6 (require "board_21343296_GerardoPerezEncina.rkt")
7 (require "game_21343296_GerardoPerezEncina.rkt")
8
9 ; 1. Creación de jugadores (10 fichas cada uno para un juego corto)
10 (define p1 (player 1 "Juan" "red" 0 0 0 10))
11 (define p2 (player 2 "Mauricio" "yellow" 0 0 0 10))
12
13 ; 2. Creación de piezas
14 (define red-piece (piece "red"))
15 (define yellow-piece (piece "yellow"))
16
17 ; 3. Creación del tablero inicial
18 (define empty-board (board))
19
20 ; 4. Creación de un nuevo juego
21 (define g0 (game p1 p2 empty-board 1))
22
23 ; 5. Realizando movimientos para crear una victoria diagonal
24 (define g1 (game-player-set-move g0 p1 0)) ; Juan coloca en columna 0
25 (define g2 (game-player-set-move g1 p2 1)) ; Mauricio coloca en columna 1
26 (define g3 (game-player-set-move g2 p1 1)) ; Juan coloca en columna 1
27 (define g4 (game-player-set-move g3 p2 2)) ; Mauricio coloca en columna 2
28 (define g5 (game-player-set-move g4 p1 2)) ; Juan coloca en columna 2
29 (define g6 (game-player-set-move g5 p2 3)) ; Mauricio coloca en columna 3
30 (define g7 (game-player-set-move g6 p1 2)) ; Juan coloca en columna 2
31 (define g8 (game-player-set-move g7 p2 3)) ; Mauricio coloca en columna 3
32 (define g9 (game-player-set-move g8 p1 3)) ; Juan coloca en columna 3
33 (define g10 (game-player-set-move g9 p2 0)) ; Mauricio coloca en columna 0
34 (define g11 (game-player-set-move g10 p1 3)) ; Juan coloca en columna 3 (victoria diagonal)
35
36 ; 6. Verificaciones durante el juego
37 (display "¿Se puede jugar en el tablero vacío? ")
38 (board-can-play? empty-board)
39
40 (display "¿Se puede jugar después de 11 movimientos? ")
41 (board-can-play? (game-get-board g11))
42
43 (display "Jugador actual después de 11 movimientos: ")
44 (game-get-current-player g11)
45
46 ; 7. Verificación de victoria
47 (display "Verificación de victoria vertical: ")
48 (board-check-vertical-win (game-get-board g11))
49
50 (display "Verificación de victoria horizontal: ")
```

Determine language from source ▼

CRLE 10 503.36 MB

Ilustración 2: Archivo "script_base_21343296_PerezEncina" en el IDE DrRacket

4.-



Ilustración 3: Botón "Run" dentro del IDE

5.-

```
Welcome to DrRacket, version 8.14 [cs].
Language: racket, with debugging; memory limit: 128 MB.
;Se puede jugar en el tablero vacío? #t
;Se puede jugar después de 11 movimientos? #t
Jugador actual después de 11 movimientos: '(2 "Mauricio" "yellow" 0 1 0 10)
Verificación de victoria vertical: 0
Verificación de victoria horizontal: 0
Verificación de victoria diagonal: 1
Verificación de ganador: 1
;Es empate? #f
Historial de movimientos: '((1 0 "red") (2 1 "yellow") (1 1 "red") (2 2 "yellow") (1 2 "red") (2 3 "yellow") (1 2 "red") (2 3 "yellow") (1 3 "red") (2 0 "yellow") (1 3 "red"))
Estado final del tablero: '((0 0 0 0 0 0 0) (0 0 0 0 0 0 0) (0 0 0 "red" 0 0 0) (0 0 "red" "red" 0 0 0) ("yellow" "red" "red" "yellow" 0 0 0) ("red" "yellow" "yellow"
"yellow" 0 0 0))
>
```

Ilustración 4: Demostración del programa funcionando correctamente