Universidad Santiago de
Chile
Facultad de Ingeniería
Departamento de
Ingeniería Informatica

Paradigmas de Programación
Informe Nº 1: "Connect 4"
«Simplificado en Scheme»

Nombre:
Alfonso Palacios Vergara

Profesor:

Gonzalo Martinez Ramirez

Fecha de entrega:

01 de Noviembre de 2024

Índice

1. Introducción

Connect 4 es un popular juego de estrategia y habilidad creado en 1974 por Howard Wexler y Ned Strongin, el cual pasaría a ser propiedad de la empresa Hasbro en 1984 y hasta la actualidad. El juego consiste de dos jugadores y un tablero vertical de 6 filas por 7 columnas. El objetivo es ser el primero en alinear cuatro fichas del mismo color, ya sea horizontal, vertical o diagonalmente.

Cada jugador elige un color de ficha y, en turnos alternos, deben dejar caer una ficha en una de las columnas del tablero. Las fichas caen hasta el fondo de la columna elegida, lo que añade un elemento de estrategia, ya que cada movimiento afecta tanto a la posición del jugador como a la del oponente. Por esta razón, ha sido utilizado en entornos educativos para desarrollar habilidades de resolución de problemas y pensamiento lógico.

Es este contexto en el cual se encuentra el contenido del presente informe, el cual tiene como objetivo presentar y detallar una implementación en lenguaje Scheme del juego Connect 4 desarrollado dentro de la idea del paradigma funcional, el cual utiliza funciones como unidad fundamental y base.

1.1. Descripción del problema

El problema abordado en este trabajo consiste en diseñar e implementar un juego de Connect 4 (o Conecta 4) de manera digital de tal modo que esté representado (por lo menos) cada juego con sus jugadores (cada uno con su nombre, color de fichas, estadísticas y fichas restantes), su tablero y su historial de juegos.

En el juego se debe permitir crear y jugar los movimientos correspondientes por cada juego, realizar la verificación de victoria, derrota o empate en el tablero y actualizar las estadísticas e información de cada jugador luego de cada movimiento.

Existen ciertas limitaciones con el problema, principalmente debido a la exigencia de desarrollar el problema utilizando el paradigma funcional. Este paradigma tiene ciertas características que presentan desafíos al momento de esctructurar y resolver el problema, como lo son la inmutabilidad y la ausencia de estados compartidos.

1.2. Descripción del paradigma

El paradigma funcional nace del cálculo lambda, una notación matemática introducida por Alonzo Church en la década de 1930. El mismo se basa en la representación de todos los cálculos por medio de funciones, las cuales se consideran 'cajas negras' que reciben elementos de entrada y devuelven elementos de salida, respetando el principio de dominio y recorrido matemático (Jung, 2004, p. 2-3). El paradigma funcional pertenece a los paradigmas declarativos, que se centran en especificar qué hay que solucionar más que en cómo solucionarlo (Jung, 2004, p. 2-3).

Es a través de esta representación que el paradigma funcional permite y restringe a sus funciones, las cuales se definen indicando sus parámetros y el procedimiento a realizar únicamente con los parámetros de entrada, al igual que una función matemática. Se hace uso de la notación prefija, anteponiendo el operador a los operandos.

Por otra parte, la currificación es una técnica fundamental en la programación funcional que permite transformar una función que toma múltiples argumentos en una serie de funciones que toman un solo argumento cada una. En lugar de recibir todos los argumentos de una vez, una función currificada recibe el primer argumento y devuelve una nueva función que espera el siguiente argumento, y así sucesivamente (Jung, 2004).

A su vez, y a diferencia de otros paradigmas que dependen de estructuras de control imperativas, la programación funcional utiliza la recursión como su principal mecanismo para la repetición y el procesamiento de datos. Las funciones recursivas se definen en términos de sí mismas, lo que permite realizar llamados recursivos para poder realizar repetidamente uno o más procedimientos expresados dentro de la misma función. Esto ayuda a expresar algoritmos de manera mas clara (Jung, 2004).

2. Desarrollo

2.1. Análisis del problema

Dados los requisitos funcionales que se deben cubrir, lo primero que se debe realizar es identificar los TDA (Tipos de Datos Abstractos) necesarios. Se identificaron 3 elementos fundamentales para todo juego de Connect 4, expuestos en la Tabla 1. La estructura sigue una dependencia jerárquica: cada juego debe poseer 1 tablero y 2 jugadores.

Con respecto a los requerimientos funcionales (ver Tabla 2), existen 4 tipos: constructor, que permite crear un tipo de dato abstracto teniendo en consideración los requisitos y tipos de datos que se requiere; modificadores, que realizan modificaciones la cantidad o el contenido de un tipo de dato abstracto; selectores, que buscan y retornan información específica dentro de diferentes tipos de dato abstractos; y otros, que se encargan de realizar operaciones que no encajan en alguno de las categorías anteriormente mencionados. De la misma manera, los archivos de TDAs están organizados con estos mismos criterios.

Por ultimo, en forma general, se puede pensar un juego de Connect 4 como un listado de información referente al tablero, jugador y juegos pasados, cada uno siendo asimismo listas con información específica a cada estructura.

2.2. Diseño de la solución

El enfoque principal de la solución es gestionar los datos a través de combinaciones de listas o pares, donde cada lista puede contener otras listas internamente *(CS5010 Program Design Paradigms Course Slides (s.f.))*. Esta estructura permite utilizar funciones primitivas de Scheme, como car, cdr y append, las cuales son fundamentales para implementar ciclos recursivos. En particular, muchas de las funciones clave aprovechan la técnica de recursión de cola, un tipo específico de recursión en el cual el llamado recursivo no deja estados pendientes, esto es, no se realiza ninguna operación con el mismo y en cambio suele utilizarse un acumulador que servirá como el elemento de salida. Se hace uso, por otra parte, de instrucciones declarativas (específicamente 'map' y 'filter') para facilitar la implementación de ciertos procesos recurrentes.

Todas estas decisiones se tomaron para hacer una implementacion únicamente basada en listas y pares. Además, se separan las estructuras general en función de lo mostrado en el Cuadro 1, que fueron separados con sus propios TDA para cada uno, agregando dentro sus funciones constructoras, modificadoras y de pertenencia. Tener las estructuras separadas de esta forma permite obtener una organización modular del código.

2.3. Aspectos de implementación

Teniendo en cuenta lo anterior, las implementaciones de los RF mencionados en la Tabla 2 se encuentran todas en un único archivo 'main', mientras

que cada TDA posee su archivo independiente organizado en funciones constructoras, modificadores, selectoras y otras.

Se realizó el código completamente en DrRacket versión 8.14 sin modificaciones externas, siendo el mismo interpretado por racket via CLI. Se utilizó un notebook HP Pavillon x360 con Windows 10x para la totalidad del proceso de programación.

2.4. Instrucciones de uso

Para utilizar los archivos se deben tener todos dentro de la misma dirección, ya que se requieren entre ellos (en específico, el archivo 'main' requiere de todos los archivos de TDA y los archivos de prueba requieren del archivo 'main'). Esto implica que ciertas funciones que están en un archivo son utilizadas en otro (en especial de TDA a main).

Los archivos de pruebas permiten visualizar los recorridos de cada función y verificar los procedimientos de las mismas teniendo en cuenta sus dominios. El mismo importa todas las funciones del archivo 'main' (RFs) y ejecuta diferentes procedimientos que sirven para probar estas mismas. Se recomienda revisar la documentación para cada uno, pues todos poseen un dominio específico con parámetros de entrada particulares. Todos los archivos se encuentran listos para ser corridos e interpretados. Asimismo, se recomienda tener conocimiento específicamente de los constructores, para tener clara la implementación utilizada para cada TDA.

2.5. Resultados y autoevaluación

En general, se pudo implementar cada RF de manera efectiva, complementándose de buena manera con los TDAs y permitiendo representar fielmente un juego de Connect 4. Se realizaron pruebas en todos los casos borde, sólo retornando algo no esperado cuando se introducía a la función algo fuera de su dominio, lo cual es aceptable siendo que cada funcion se especializa en un tipo de operación y nunca serán llamadas con algo indebido (y de todas formas es responsabilidad del usuario velar por que esto no pase). El resultado de la autoevaluación tiene en su totalidad puntaje máximo en todos los ítem excepto en aquellos que sé que pude haber implementado mejor, dado más tiempo.

3. Conclusiones

Los alcances de la implementación se consideran aceptables para una buena representación digital en Scheme de Connect 4. Lo más difícil fué el cambio de paradigma y acostumbrarse a programar de forma no imparativa, así como aprender las funciones fundamentales de Racket Scheme. También hubieron varias ocasiones en las que se debió re-hacer una o varias funciones debido a errores en el entendimiento de sus requerimientos, lo que contribuyó al tiempo y a la frustración en la realización de la implementación. Fue crucial pasar mas tiempo interpretando lo que se pide mas que programándolo, fue de mucha utilidad hacer los requerimientos de manera inversa, del detalle mas chico hasta la totalidad de la función.

Referencias

[1] Cs5010 program design paradigms course slides. (s.f.). https://course.ccs.neu.edu/cs5010sp16/Slides/. (Accessed: 2024-10-31)

[2] Jung, A. (2004). A short introduction to the lambda calculus. University of Birmingham, School of Computer Science. https://www.cs.bham.ac.uk/ axj/pub/papers/lambda-calculus.pdf (Accessed: 2024-10-31)

[3] Cs345 - an introduction to scheme and its implementation. (1997, January). https://www.cs.utexas.edu/ftp/garbage/cs345/schintro-v14/schintrotoc.html. (Accessed : 2024-10-31)

[4] Flatt, M. PLT. (s.f.) - The Racket Reference. https://docs.racket-lang.org/reference/ (Accessed : 2024-10-31)

4. Anexos

Elemento	**Descripción**
Game	Juego de Connect 4 compuesto por 2 jugadores, un tablero y un historial de juego
Board	Tablero de juego de Connect 4 compuesto por filas y columnas, con 42 espacios totales
Player	Jugador de un juego de Connect 4 compuesto por su id, nombre, el color de sus piezas, cantidad de victorias, derrotas y empates, y piezas restantes

Cuadro 1: Elementos principales Connect 4

RF	Descripción
1	TDAs - Realizar TDAs para cada estructura relevante en un archivo separado cada uno, siguiendo estrucciones específicas
2	TDA Player - constructor. Función que permite crear un jugador.
3	TDA Piece - constructor. Función que crea una ficha de Conecta4.
4	TDA Board - constructor. Crear un tablero de Conecta4.
5	TDA Board - otros - sePuedeJugar?. Función que permite verificar si se puede realizar más jugadas en el tablero.
6	TDA Board - modificador - jugar ficha. Jugar una ficha en el tablero
7	TDA Board - otros - verificar victoria vertical. Función que permite verificar el estado actual del tablero y entregar el posible ganador que cumple con la regla de conectar 4 fichas de forma vertical.
8	TDA Board - otros - verificar victoria horizontal. Función que permite verificar el estado actual del tablero y entregar el posible ganador que cumple con la regla de conectar 4 fichas de forma horizontal.
9	TDA Board - otros - verificar victoria diagonal. Función que permite verificar el estado actual del tablero y entregar el posible ganador que cumple con la regla de conectar 4 fichas de forma diagonal.
10	TDA Board - otros - entregarGanador. Función que permite verificar el estado actual del tablero y entregar el posible ganador que cumple con la regla de conectar 4 fichas de forma diagonal
11	TDA Game - constructor. Función que permite crear una nueva partida
12	TDA Game - otros - history. Función que genera un historial de movimientos de la partida.
13	TDA Game - otros - esEmpate?. Función que verifica si el estado actual del juego es empate
14	TDA Player - otros - actualizarEstadisticas. Función que actualiza las estadísticas del jugador, ya sea victoria, derrotas o empates.
15	TDA Game - selector - getCurrentPlayer. Función que obtiene el jugador cuyo turno está en curso.
16	TDA Game - selector - board-get-state. Función que entrega el estado actual del tablero en el juego.
17	TDA Game - modificador - game-set-end. Función finaliza el juego actualizando las estadísticas de los jugadores según el resultado.
18	TDA Game - modificador - realizarMovimiento. Función