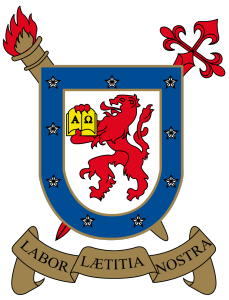

Laboratorio 1
Paradigmas de programación

Programación Funcional - Scheme Facultad de Ingeniería -

Universidad de Santiago de Chile

Estudiante: Mateo Valle Lacourt
Profesor: Gonzalo Martinez Ramirez

Índice

1. Descripción breve del problema

El problema presentado en este laboratorio fue realizar un juego de Conecta-4 en el lenguaje de programación Scheme-Racket, bajo el paradigma de programación funcional. Agregando variedad las características del juego, tal como:

- Las fichas deben caer en lo más bajo del tablero.
- Verificar si hay cuatro fichas seguidas de forma horizontal, vertical y diagonal.
- Actualizar estadísticas de los jugadores involucrados en el juego.

El desafío del laboratorio fue comprender a usar este nuevo paradigma y hacer que todos nuestros conocimientos se adapten a este. También incluir el uso de las nuevas herramientas de trabajo presentadas para la realización del laboratorio, tales como: Git-GitHub y la implementación de IA para jugar Conecta-4.

2. Descripción breve del paradigma

El origen del Paradigma Funcional se sitúa en hace más de 50 años, este se basa en el sistema del Calculo Lambda, propuesto por Alonzo Church, el sistema presenta una formalización a la idea de función, la cual tiene una entrada y salida (importante detallar que la salida de una función sólo depende de la entrada).

El Paradigma Funcional utiliza el concepto de Cálculo Lambda para representar problemas del mundo real en base de funciones, las cuales son su unidad de abstracción elemental. El paradigma también utiliza algunas de las aplicaciones de las funciones, tales como la recursión, currificación, uso de funciones anónimas, etc.

Esta representación tiene virtudes y desventajas a la hora de programar, por una parte, gracias a su nivel de abstracción, logra que el código sea más simple y legible. Por su contraparte, debido a su gran limitación, es imposible el uso de variables mutables, por lo que solo se pueden representar mediante un estado estático. También puede que su implementación no sea las más práctica a la hora de representar problemas del mundo real.

3. Análisis del problema

Dados unos requisitos funcionales, se debió analizar que TDA's usar, para la representación de juego en este proyecto se utilizaron cuatro TDA's: jugadores, piezas, tablero y estados de juego. Todos ellos representados en base de listas/ pares. Estos tipos de datos están relacionados mediante funciones y se comunican entre sí para que el sistema de juego funcione, por ejemplo:

- El jugador guarda una pieza.
- El tablero guarda dos jugadores y una representación de tablero.
- El juego guarda los jugadores y un tablero, incluyendo un historial.

Los TDA's usados en el proyecto tienen distintos tipos de funciones, las cuales se categorizan de la siguiente manera:

- Funciones constructoras: se usan para crear un "objeto" del TDA.
- Funciones selectoras: retornan un dato específico del TDA.
- Funciones modificadoras: permiten crear un nuevo estado del TDA con algún dato modificado.
- Funciones otras: las cuales realizan tareas relacionadas con el contexto de funcionamiento del TDA con el proyecto en general.

En esta implementación no se usaron funciones de pertenencia, ya que no fueron necesarias.

Finalmente, el juego se estructura por:

- Jugadores, los cuales contienen sus piezas y su cantidad.
- Funciones selectoras: retornan un dato específico del TDA.
- Estado de juego, contiene jugadores, tablero, contador de turno para que las jugadas sean correctas.

Cabe destacar que, cuando se detecte alguna victoria en el tablero, las estadísticas de jugador se actualizan automáticamente y queda guardado en ultimo estado de juego, debido a la inmutabilidad de variables, si se quiere jugar otro juego, se deben crear nuevos estados de jugador. Obteniendo la información del juego anterior o cambiando las estadísticas manualmente.

4. Diseño de solución

En las primeras etapas de solución, se requirió implementar los TDA's, llegando a la conclusión de que lo mejor sería representarlos mediante listas/pares, para almacenar y recuperar información eficientemente.

Lo primero que se construyó fueron las funciones constructoras y selectoras de los TDA's.

Luego se emplearon las funciones del tablero, para colocar piezas y detectar victorias en el tablero, las cuales se realizaron con recursividad Natural y por Cola, estas siendo vistas en clases. En las primeras etapas del proyecto, no se realizaron de esta manera, ya que la prioridad era terminar el código de una forma general, pero en las últimas partes de diseño, se reestructuraron estas y otras funciones para que cumplan con los requisitos pedidos.

Una vez terminadas las funciones del tablero, se empezaron a implementar las funciones del TDA juego, como la función crear historial, la cual fue un tema complejo, ya que no se tuvo en cuenta en la primera implementación, por lo tanto, se creó una segunda función constructora. Finalmente se implementó la función colocar pieza, hecha a partir de la mayoría de las funciones que fueron creadas, por lo que su implementación declarativa fue muy fácil de hacer, ya que las funciones para llamar ya estaban hechas.

Además, incluir de que en la última fase de diseño se optimizaron e implementaron de una forma más estructurada y ordenada las funciones.

Para la creación del proyecto, se consultaron exclusivamente tres fuentes, las cuales son:

- Clases presenciales y Videos de Uvirtual.
- Documentación oficial de Racket. [1]
- Lista de reproducción: Aprendizaje de racket. [2]

5. Aspectos de implementación

La totalidad del proyecto fue hecho en el programa DrRacket versión 8.14, y la única librería utilizada fue la predeterminada del programa. El proyecto se estructura en cuatro archivos principales, los cuales contienen los TDA player, piece, board y game con sus respectivas funciones. Estos archivos son vitales, ya que dependen entre sí, por lo que siempre hay que tenerlos en la misma dirección.

6. Instrucciones de uso

La entrega cuenta con un archivo de pruebas, el cual contiene tres ejemplos de uso de cada función requerida en el enunciado del laboratorio, igualmente es recomendable ver los archivos de TDA y leer que hace cada función, estas cuentan con información sobre que es lo que hace, sus dominios y recorridos, y el tipo de recursión utilizado.

Para poder jugar Conecta-4 se debe crear un archivo ".rkt", importando los archivos mencionados. Sin embargo, la entrega cuenta con Scripts, con simulaciones de juego para poder entender bien como jugar. En dichos Scripts, hay ejemplos de jugadas de todo tipo:

- Victoria horizontal, vertical y diagonal.
- Empate.
- Jugar dos juegos seguidos.
- Jugar en un turno no correspondiente.
- Jugar en una columna no correspondiente.

Posibles errores:

La implementación no limita a los jugadores a que elijan el mismo color, por lo que, si lo hacen, o el color de los jugadores empieza con la misma letra, habrá problemas para representar el historial, el tablero, y la forma de descubrir si hay ganador en el juego. Pero, según las reglas de Conecta-4, se deberían usar fichas rojas y amarillas, por lo que si se sigue este formato no habrá ningún problema.

7. Resultados y autoevaluación

Se obtuvo un proyecto exitoso y funcional en su totalidad, se lograron respetar todos los requerimientos funcionales, con sus requisitos específicos, por lo que la autoevaluación considera el puntaje máximo de cada requerimiento. También agregar que se trabajó constantemente en el proyecto, logrando, casi un commit diario en el repositorio.

8. Conclusiones del trabajo

Para concluir, se puede decir que el proyecto fue un real desafío, ya que lo más complicado fue abstraerse, darle una buena implementación al proyecto, pensar de una nueva forma y poder entender el uso del nuevo paradigma, teniendo en cuenta que no hay variables mutables. Añadir también la dificultad de implementar tipos de recursión específicos, sean estos los Natural y Cola. Sin embargo, gracias a la constancia, y al tipo de trabajo, el cual es cercano a la vida laboral, se logró un proyecto exitoso y se adquirió un nuevo conocimiento, el cual puede ser muy útil para el futuro laboral y académico.

9. Referencias

Referencias

[1] R. Team, "Racket documentation," s.f, accedido el 28 de octubre de 2024. [Online]. Available: https://docs.racket-lang.org/reference/index.html

[2] Makigas, "Aprendizaje de racket," 2014, accedido el 28 de octubre de 2024. [Online]. Available: https://www.youtube.com/playlist?list=PLTd5ehIj0goMswKV4Glhr4uixrhCmihIt