Universidad De Santiago De Chile
Facultad de ingeniería
Departamento de ingeniería informática

Informe de Laboratorio: Juego Dobble

Nombre: Cristobal Francisco Marchant Osorio
Profesor: Roberto Gonzales I.
Asignatura: Paradigmas de programación

17 de Abril del 2022

Tabla de contenidos

Pagina contenido

Introducción

Este informe habla sobre el desarrollo del laboratorio N°1 de paradigmas de programación, este consiste en que gracias al compilador Dr. Racket, podemos usar el lenguaje de programación Scheme. En este informe se plantean los objetivos de explicar brevemente el problema planteado, darle un análisis a este problema, generar un diseño a la solución de este, mostrar aspectos de la implementación, instrucciones y como hacer usos de estas, mostrar resultados, una autoevaluación y una conclusión acerca del trabajo.

Descripción del problema

El problema planteado consiste en desarrollar el juego de cartas “Dobble”, en donde cada carta tiene un elemento en común. Sobre este tema tenemos 2 partes, en la primera parte se pide implementar la creación de un mazo y poder manipular este mazo para poder obtener valores como el total de cartas, si un mazo es válido, encontrar una carta en cierta posición, y en la segunda parte se nos pide crear una manera en donde el usuario pueda jugar con estas cartas, el usuario puede registrarse, elegir un modo de juego y elegir un mazo, además implementar funciones que permitirá a los jugadores poder pasar el turno, jugar al juego y finalizar el juego.

Descripción del paradigma

En este trabajo se trabajará bajo los estándares del paradigma funcional, el cual se basa en la abstracción y una forma de trabajar de manera declarativa.

Teniendo en cuenta la definición podemos obtener 2 preguntas ¿qué es un paradigma? y ¿Qué es una función?, obviamente ambas preguntas aplicadas a la programación funcional. Entonces ¿Qué es un paradigma?, el paradigma es un modelo a seguir o una rutina la cual se basa en el trabajo con lambda, la cual se basa principalmente en el que ¿Qué? y no en ¿Cómo?, es decir, este no ve cómo se obtuvo el resultado, sino que este se enfoca en el resultado mismo.

El paradigma funcional trabaja junto a las funciones, lo cual nos lleva la pregunta, ¿Que es una función?, una función es un bloque de programación el cual tiene el objetivo de realizar una operación, esto con el fin de dar respuesta o expresar algo, esta se compone por elementos de entrada y en la mayoría de los casos elementos de salida. En la programación funcional como se mencionó anteriormente trabaja junto a las funciones, por lo que no hay usos de variables actualizables, de las cuales se desprenden distintos usos, tal es el caso de la currificación la cual consiste en una función de 2 elementos que devuelve el valor, está destaca por manera de ser implementada. También cabe destacar tipos de funciones existentes, tales como las funciones anónimas y las funciones de orden superior, las funciones anónimas son funciones que trabajan bajo un parámetro de entrada y una transformación para esta función, y por su parte las funciones de orden superior son funciones que tienen como parámetro de entrada una función, un ejemplo es el uso de “map” en el lenguaje de scheme.

Sobre la implementación en el lenguaje de programación Scheme existe un estilo en específico que es fundamental para trabajar en este lenguaje y fundamental en el trabajo que se está realizando en este informe, esta es la recursividad, la cual puede realizar un sin fin de operaciones en donde se evalúan casos, se obtienen resultados, etc. Estas pueden ser de 3 tipos distintos el tipo cola, tipo natural y tipo arbórea.

Desarrollo

Análisis del problema

El desarrollo de esta actividad consta en la construcción de 2 TDAs, el TDA para la creación y modificación de cartas, y en el TDA sobre la creación del juego, siendo el primero de este tda recurrido por el segundo, es por esto que se dispone a explicar el problema en donde podamos analizar los datos de entrada y salida para la base del constructor

TDA cardsSet

Para entender la base del problema es necesario tener en cuenta las definiciones para las variables, por ejemplo, para la creación de las cartas se ingresa un número de elementos por carta, pero este número es un n+1, el n tiene que cumplir condiciones para que este sea válido, como es el caso de que este sea primo, porque si no fuera primo existirían cartas con más de 2 elementos en común o que no tengan ningún elemento en común y esto no es posible, ya que por la definición de las reglas del juego, cada carta tiene que tener un solo elemento en común.

Entonces tenemos las siguientes funciones con los siguientes parámetros de entrada y salida:

Constructor	Lista de elementos, cantidad de elementos, máximo de cartas y función de azar	Mazo de cartas
¿Es un mazo valido?	Mazo de cartas	Verdadero o falso
Numero de cartas	Mazo de cartas	Número total de cartas
Una carta en cierta posición	Mazo de cartas, la posición	Una carta
El total de cartas	Una Carta	El total de cartas
El total de elementos	Una carta	El total de elementos
De un mazo recortado encontrar las cartas restantes	Un mazo	Mazo con las cartas faltantes
Transformar el set de cartas a un string	Un mazo	string

TDA Game

Para entender la base de este problema es necesario tener la función de la creación de un mazo, este se basa en el supuesto de que el mazo ingresado es un mazo valido y que la cantidad de cartas sean suficiente para la cantidad de jugadores según el modo de juego empleado.

Entonces tenemos las siguientes funciones con los siguientes parámetros de entrada y salida:

Función	Entrada	Salida
Constructor de la mesa	Cantidad de jugadores, el mazo de cartas, modo de juego y una función de azar	La mesa en donde se desarrolle el juego
Modo de juego StackMode	Mazo	Implementación de como usarlo
Registro de jugador	Mesa y el nombre del jugador	La mesa con el jugador incluido
De quien es el turno	Mesa	El nombre del jugador de turno
Permite jugar	Mesa y una función (mostrar mesa, pasar turno, entregar un resultado y terminar el juego con tabla de posiciones)	Mesa actualizada según la función aplicada
Status	Mesa	Entrega el estado actual de la mesa
Puntuación	Mesa y un nombre	Entrega la puntuación de tal persona
Mostrar la mesa como string	Mesa	La mesa como string

Otras funciones

También está la creación de funciones adicionales tal es el caso de agregar cartas, otros modos de juegos, etc.

Diseño de solución

Para plantear el diseño de la solución aplicada en este trabajo, se usarán 2 tipos de recursión, la natura y la de cola, y se trabajara en función de las listas.

TDA cardsSet

A continuación, se explicarán como funcionarán las entradas para el mazo de cartas, para la creación de este mazo se tomarán en cuenta 4 aspectos de entrada:
El primero es la lista de elementos, esta fue planteada para que cuando se tenga un mazo de números se haga una comparación entre el número y la posición del elemento en la lista de elementos y así los vaya agregando a una nueva lista (cabe aclarar que la lista está definida al inicio de archivo y esta puede ser modificada, pero solo trabaja con strings).
El segundo parámetro de entrada es el número de elementos por carta, este aplico de manera (((NumeroElementos - 1)^2) + NumeroElementos) es así que podíamos saber cuántas cartas tendríamos en total.
El tercer parámetro fue el máximo de cartas, la implementación fue pensada de manera en que cuando ya se tuviera el mazo completo, las cartas se vayan agregando a una lista nula hasta que el número máximo sea igual a 0.
El cuarto parámetro fue de cómo desordenar el mazo, es por esto que se implementó una función que generaba un numero al azar con máximo al largo del número de cartas, luego encuentra ese número lo agrega a una lista nueva, lo elimina de la antigua y se genera una recursión, tal y como se muestra en la imagen 1, en donde el parámetro de entrada fuera el mazo y un true o un false. Si era true se generaría un mazo al azar, y si fuera false el mazo se quedaría tal y como esta. La implementación del random se muestra a continuación: (ver anexo, figura 1)

Para el caso de “Dobble?” este se pensó en definir una función distinta para cada condición, estos son, que cada carta tenga la misma cantidad de elementos, que cada carta solo tenga 1 elemento en común con otra carta (esto también comprueba si el número de elementos por carta era válida) y por último que una carta tuviera elementos distintos, cuando se tuvieron estas funciones se usaron para crear una función mayor en donde si las 3 eran true, el resultado del “Dobble?” es true, en cualquier otro caso el resultado será false.

El numCards se pensó simplemente en un length del mazo, pero en el trabajo se implementó un propio length que tenía un acumulador y cuando la lista fuera vacía devuelve lo del acumulador.

La NthCard esta se pensó es ir recorriendo la lista con recursión y cdr de la misma lista, y por cada carta que pase se le quite 1 a la posición, entonces cuando este llegue a 0, regrese el car de la lista.

En el caso de FindTotalElements se pensó en contar la cantidad de elementos de la carta y aplicar la fórmula de ((n-1)**2)+n.

En requiereElements se pensó en la misma función del punto anterior, ya que el total de elementos es equivalente al total de las cartas

En el missingCards se pensó de modo en comprar el mazo entregado con una copia completa de este mismo, entonces de este modo si la carta no se encuentra en el mazo entregado, la agrega en una nueva lista, así sucesivamente hasta que la lista quede vacía y retorne la lista de las cartas faltantes.

En el CardSet->String se pensó de manera en que primeramente una carta se transforme en un string y se agregue a una lista, luego cada carta que se convierto en string se le agregan parámetros estéticos, y para finalizar la lista con las cartas en string y co los parámetros estéticos los juntamos todos en un solo string.

El addCards se pensó de manera se agregará a la lista la carta, ¿se compruebe con “Dobble?” si el nuevo mazo es válido, si es válido se agrega, y si el mazo es no valido regresa el mazo original sin agregar la nueva carta

TDA Game

Para la creación del constructor del constructor del game se pensó en definir una lista que funcionará de mesa de juego en donde este incluirá los datos de cada jugador (definidos por una función player (inicialmente es un número, turno = 0 y puntaje = 0)) y dependiendo del modo de juego, se podrán ver las cartas en juego y las cartas restantes.

Para la creación del modo de juego StackMode se pensó en una lista de 2 elementos, el primer elemento una lista de las cartas volteadas y el segundo elemento el resto de las cartas.

Para la creación del register se baso en la definición de player, si el primer elemento era un numero se agregaba el nombre, si el primer elemento era un string y los nombres no coincidían pasaba al siguiente player no registrado, si el nombre ya estaba registrado no lo agregaba.

Para la creación del Selector de quien es el turno, esta se baso en buscar quien estaba de turno según la lógica que la lista siempre vaya de izquierda a derecha y cuando llegaba al final volvía al primero, entonces cuando la comparaba a 2 usuarios del juego se veía cual era mayor, entonces el menor de estos 2 estaba de turno, pero si eran iguales se compararán el segundo con el tercero y si todos son iguales el turno seria del primero de la lista.

Para la creación del play y la implementación de sus funciones en action se pensó en que si el action era null, este pasaría a la función “Nulo” y devolvería la partida, luego en el spotIt se pensé en que si el elemento que entregaba era coincidía con ambas cartas en el mazo este era true, entonces se sumaria un turno y se agregaría un punto, en caso de que estuviera malo seria false y solo pasaría el turno, luego la función pass simplemente como es un modificador aumento en 1 el turno del jugador en turno, y por último la función finish hace una comparativa entre los puntajes y determina al ganador con el numero 1, al

segundo con el número 2, así sucesivamente hasta que estén todos los jugadores contemplados y elimina todas las cartas de la partida y da el juego por terminado.
El diagrama de cómo funciona la mesa de juego junto a otras funciones establecida hasta este punto se muestra a continuación: (ver anexo, figura 2)

Para la función status esta al ser un selector solo devolvía la mesa actual.
El score también este siendo un selector buscaba el nombre de la persona ingresada y devolvía su puntaje actual.

Las ultimas funciones no fueron implementadas.

Aspectos de implementación

En el siguiente apartado se darán a mostrar aspectos implementados que son relevantes a la hora de entender el desarrollo del trabajo.
Cabe destacar que como aspecto de mi implementación realiza gran parte del trabajo a base de “cond” en vez de “if”, ya que encontré mas comodo de trabajar.

Estructura del proyecto

El proyecto presenta una estructura que se compone de 2 TDAs, y un archivo main en donde se incluyen los ejemplos.

TDA - CardsSet: Es el encargado de la generación del mazo de cartas que cumple con los parámetros ingresados por el usuario, además de incluir funciones de pertenencias, modificadores, selectores y otras.

TDA - Player: Pequeño TDA definido para la creación del perfil del usuario compuesto por un numero y acompañado de 2 ceros, en donde uno representa el turno y el otro el puntaje.

TDA - Game: Es el encargado de generar una mesa de juego en donde un usuario se pueda registrar y jugar, cabe destacar que este implementa función del TDA – CardsSet y del TDA – Player, también que al igual que el primer TDA, esta implementa funciones de pertenencias, modificadores, selectores y otras.

Bibliotecas empleadas

No se hizo uso de alguna biblioteca adicional.

Compilador

DrRacket, version 8.4

Instrucciones de uso

Existen un gran numero de ejemplos por cada función principal establecida, estos ejemplos se pueden observar en el archivo “Main”, de igual manera se mostrarán algunos ejemplos que sean comparables y que cuenten con una breve descripción y resultados esperados.

(define CARTA (cardsSet ListS 4 3 #true)) teniendo en cuenta que “ListS” es una lista de símbolos con la suficiente capacidad de generar el set de cartas. Esto generaría un Set de cartas de 4 cuatro elementos por carta y solamente 3 cartas, las cuales estarán desordenadas.

(define CARTA2 (cardsSet ListN 3 -1 #true) suponiendo que “ListN” es una lista de símbolos con solo 5 elementos. Esto generaría un set de cartas de 3 elementos por cartas y retornara todas las cartas desordenadas. En este caso el conjunto al faltarles elementos los lugares que no abarca la “ListN” serian reemplazados por números.

(define CS->S1(CardsSet->String(cardsSet null 4 -1 #true))) o (define CS->S1(CardsSet->String(cardsSet ListS 4 -1 #true))) entregaría un string del mazo de cartas con los parámetros de ordena aleatorio, 4 elementos por carta y mostrar todas las cartas, el resultado esperado sería así:
(ver anexo, figura 3)

Resultados y autoevaluación

A partir de los resultados obtenidos y poniendo a prueba los ejemplos propuestos en el archivo “Main” se cumple con éxito la mayoría de las funciones.
A continuación, se presenta la autoevaluación.

Elementos por evaluar	Calificación
TDAs	1
TDA cardsSet - constructor	1
TDA cardsSet - dobble?	1
TDA cardsSet - numCards	1
TDA cardsSet - nthCard	1
TDA cardsSet - findTotalCards	1
TDA cardsSet - requiredElements	1
TDA cardsSet - missingCards	1
TDA cardsSet - cardsSet->string	1
TDA game - constructor	1
TDA game - stackMode	1
TDA game - register	1
TDA game - whoseTurnIsIt?	1
TDA game - play	0,75
TDA game - status	1
TDA game - score	1
TDA game - game->string	0
TDA cardsSet - addCard	1
TDA game - emptyHandsStackMode	0
TDA game - myMode	0

Conclusión

Luego de terminar el proyecto se puede concluir que se cumplió la mayoría de los objetivos planteados a excepción del “game->string”, el “emptyHandsStackMode” y el myMode. Con respecto a los logros del proyecto se puede afirman que se cumplieron con éxito y se trabajo de una manera ordenada y bien documentada, una de las complicaciones del trabajo fue no tener una idea clara de cierta función, en donde a base de un análisis se pudo llegar a comprende e implementar, de igual manera otra complicación fue el tiempo por el cual deje de lado algunas funciones por no poder completarlas, ya que para cada fucnion o TDA se necesita un gran tiempo de comprensión e implementación.

Sobre DR Racket no hubo complicaciones a excepción de la generación de ciclos entre funciones, pero eso fue un error personal que se pudo solucionar y sobre la plataforma git no hubo ningún inconveniente solo que no se pudo hacer commits diarios por la carga academia que se tiene aparte de este ramo.

Se espera que para próximos trabajos trabajar con mas tiempo, planear la idea de la implementación desde antes y trabajar de manera más ordenada.

Bibliografía

Álvaro Rojas, 2020. ¿Qué es la programación funcional?.
https://www.incentro.com/es-ES/blog/que-programacion-funcional

Roberto Gonzales, 2022 Proyecto semestral de laboratorio.
https://docs.google.com/document/d/1tCVZBGScJbXFspQfu6WlDl3PsyCJUcqIgos5__DQz7k/edit#

Roberto Gonzales, 2020 Material online Mostrado en clases.

Anexo

```
;-----------------------------------------RANDOM-----------------------------------------------------------------------------------
;Funcion encotrar
;Dominio: Lista X Entero
;Recorrido: Lista
;Descripcion: recorre la lista hasta encontrar el elemento por su posicion y lo devuelve
;Tipo: Cola
(define (encontrar L n)
  (cond
    [(empty? L) null]
    [(equal? n 1)(car L)]
    [else (encontrar (cdr L) (- n 1))]))

;Funcion eleminar
;Dominio: Lista X Entero
;Recorrido: Lista
;Descripcion: Recorre la lista hasta encontrar un elemento por su posicion y lo borra
;Tipo: Natural
(define (eleminar L n)
  (cond
    [(empty? L) null]
    [(equal? n 1)(cdr L)]
    [else (cons (car L) (eleminar (cdr L) (- n 1)))]))

;Funcion randomFn
;Dominio: Lista X Entero
;Recorrido: Lista
;Descripcion: desordena elementos de una lista
;Tipo: Natural
(define (randomFn L NR)
  (cond
    [(equal? (length L) 1) (cons (car L) null)]
    [else (cons (encontrar L NR) (randomFn (eleminar L NR) (random 1 (+(length (eleminar L NR))1)))))]))
```

Figura 1, Función random.

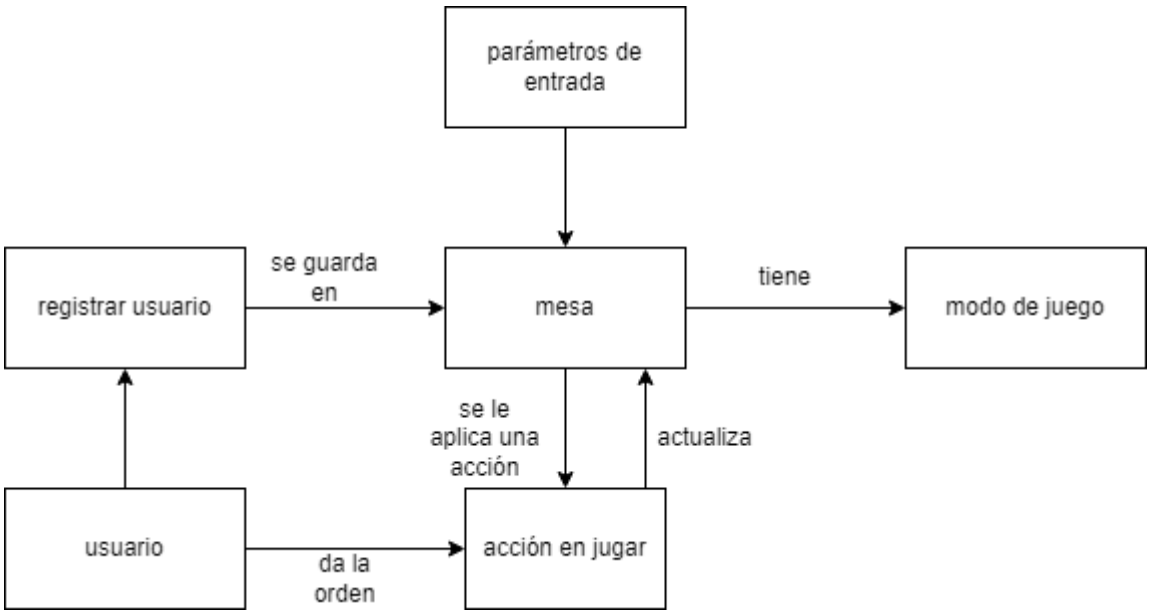

Figura 2, Diagrama del funcionamiento del game.

```
"Card 1: B F I L \nCard 2: A H I J \nCard 3: D E J L \nCard 4: A B C D \nCard 5: A K L
M \nCard 6: B G J M \nCard 7: B E H K \nCard 8: C E I M \nCard 9: D G I K \nCard 10: C
G H L \nCard 11: C F J K \nCard 12: A E F G \nCard 13: D F H M \n"
```

Figura 3, CardsSet->String.