Universidad de Santiago de Chile
Facultad de Ingeniería
Departamento de Ingeniería en Informática

Informe 1 de Laboratorio: Paradigma Funcional

Nombre: Leonardo Espinoza Ortiz
Profesor: Guillermo Truffa
Asignatura: Paradigmas de Programación
Fecha de entrega: 26-09-2022

Tabla de contenidos:

Introducción:

En el presente informe se presentará un laboratorio acerca de uno de los Paradigmas vistos en clases de Paradigmas de programación, el Paradigma Funcional.
Este Paradigma Funcional será practicado mediante el lenguaje de programación Scheme a través del compilador llamado DrRacket.
Este trabajo tendrá una estructura en donde se dará a conocer el problema a resolver, un análisis, diseños de la solución y finalmente una conclusión.

Descripción breve del problema:

Se pide implementar en el Lenguaje Scheme una aplicación de edición de imágenes similar a herramientas conocidas como GIMP y Adobe Photoshop.

Un software de edición o manipulación de imágenes digitales es aquel que permite a un usuario realizar distintas operaciones sobre éstas.
Existen distintos tipos de operaciones tales como rotar una imagen, invertirla, rotarla, retocarla, transformarla y redimensionarla, entre otras.
Este software se concentrará en trabajar en imágenes RGBD.

Descripción breve del paradigma:

El paradigma funcional, tal como dice su nombre, se basa en programación orientada al uso de funciones.
Entre sus principales características, la función es la unidad básica y está basado en funciones matemáticas(inyectividad).
También, no existe el concepto de variables actualizables, no existen las iteraciones y no existe el concepto de estado.

Algunos elementos importantes de este paradigma son la currificación, funciones anónimas, recursividad, funciones de orden superior entre otras.

Análisis del problema:

De forma particular, se consideran 3 tipos de representaciones para la implementación de las imágenes.
-bitmaps-d para imágenes donde cada pixel o pixbit puede tomar el valor 0 o 1

-pixmap-d para imágenes donde cada pixel o pixbit es una combinación de los valores para los canales R, G y B

-hexmaps-d para imágenes donde cada píxel o pixhex expresa la información del color del píxel a través de un valor único hexadecimal de 6 valores.

Diseño de la solución

Para construir un diseño para esta solución, se implementarán TDAs(tipos de datos abstractos) que deben poseer la estructura de Representación, Constructores, Pertenencia, Selectores, Modificadores y otras funciones.
Estos TDA servirán para una correcta y ordenada estructura de esta solución.

Estos son los TDA más importantes a considerar:

-**TDA image - constructor**

Función constructora de imágenes con bitmaps, hexmaps o pixmaps que incluye información de profundidad en cada pixel.

Dominio: Width (int) X Height (int) X [pixbit-d* | pixrgb-d* | pixhex-d*]

Recorrido:image

Ejemplo: (image 2 2 (pixbit-d 0 0 1 10) (pixbit-d 0 1 0 20) (pixbit-d 1 0 0 30) (pixbit-d 1 1 1 4))

-TDA image - bitmap?

función que permite determinar si la imagen corresponde a un bitmap-d.

Dominio: image

Recorrido:boolean

Ejemplo:(bitmap? (image 2 2 (pixbit-d 0 0 1 10) (pixbit-dpixbit 0 1 0 20) (pixbit-dpixbit 1 0 0 30) (pixbit-dpixbit 1 1 1 4)))
;result: #t

-TDA image - pixmap?

función que permite determinar si la imagen corresponde a un pixmap-d.

Dominio: image

Recorrido:boolean

Ejemplo:(pixmap? (image 2 2 (pixbit-d 0 0 1 10) (pixbit 0 1 0 20) (pixbit 1 0 0 30) (pixbit 1 1 1 4)))
;result: #f

-TDA image - hexmap?

función que permite determinar si la imagen corresponde a un hexmap-d.

Dominio: image

Recorrido: boolean

Ejemplo: (hexmap? (image 2 2 (pixbit-d 0 0 1 10) (pixbit 0 1 0 20) (pixbit 1 0 0 30) (pixbit 1 1 1 4)))
;result: #f

-TDA image - compressed?

función que determina si una imagen está comprimida.

Dominio:image

Recorrido:boolean

Ejemplo:(compressed? (image 2 2 (pixbit-d 0 0 1 10) (pixbit 0 1 0 20) (pixbit 1 0 0 30) (pixbit 1 1 1 4)))
;result: #f

-TDA image - flipH

función que permite invertir una imágen horizontalmente.

Dominio: image

Recorrido: image

Ejemplo:(flipH (image 2 2 (pixbit-d 0 0 1 10) (pixbit 0 1 0 20) (pixbit 1 0 0 30) (pixbit 1 1 1 4)))

-TDA image - flipV

función que permite invertir una imágen verticalmente.

Dominio: image

Recorrido: image

Ejemplo: (flipV (image 2 2 (pixbit-d 0 0 1 10) (pixbit 0 1 0 20) (pixbit 1 0 0 30) (pixbit 1 1 1 4)))

- TDA image - crop

Recortar una imágen a partir de un cuadrante.

Dominio:*image X x1 (int) X y1 (int) X x2 (int) X y2 (int)*

Recorrido:image

Ejemplo:(crop Img1 10 10 40 40)

;donde Img1 es una imagen creada con image

-TDA image - imgRGB->imgHex

Transforma una imagen desde una representación RGB a una representación HEX.

Dominio: image

Recorrido: image

Ejemplo: (imgRGB->imgHex (image 2 2 (pixrgb-d 0 0 10 10 10 10) (pixrgb-d 0 1 20 20 20 20) (pixrgb-d 1 0 30 30 30 30) (pixrgb-d 1 1 40 40 40 40)))

- TDA image - histogram

Retorna un histograma de frecuencias a partir de los colores en cada una de las imágenes.

Dominio:image

Recorrido:histogram

Ejemplo:(histogram (image 2 2 (pixbit-d 0 0 1 10) (pixbit 0 1 0 20) (pixbit 1 0 0 30) (pixbit 1 1 1 4)))

- TDA image - rotate90

rota la imágen 90° a la derecha

Dominio: image

Recorrido: image

Ejemplo: (rotate90 (image 2 2 (pixbit-d 0 0 1 10) (pixbit 0 1 0 20) (pixbit 1 0 0 30) (pixbit 1 1 1 4)))

-TDA image - compress

Comprime una imágen eliminando aquellos pixeles con el color más frecuente

Dominio:image

Recorrido: image

Ejemplo: (compress (image 2 2 (pixrgb-d 0 0 10 10 10 10) (pixrgb-d 0 1 20 20 20 20) (pixrgb-d 1 0 30 30 30 30) (pixrgb-d 1 1 40 40 40 40)))

-TDA image - edit

Permite aplicar funciones especiales a las imágenes. Por ejemplo, para modificar colores en alguno de los canales, pasar a blanco y negro, etc.

-TDA image - invertColorBit

Función que permite obtener el valor del bit opuesto.

-TDA image - invertColorRGB

Función que permite obtener el color simétricamente opuesto en cada canal dentro de un píxel.

-TDA image - adjustChannel

Función que permite ajustar cualquier canal de una imagen con pixeles pixrgb-d, incluido el canal de profundidad d

-**TDA image -** image->string

Función que transforma una imagen a una representación string.

-**TDA image -** depthLayers

Función que permite separar una imágen en capas en base a la profundidad en que se sitúan los pixeles.

-**TDA image -** decompress

Función que permite descomprimir una imágen comprimida

Aspectos de implementación

Para aspectos de implementación, el compilador será Dr.Racket. en su versión 8.6

El código tendrá un archivo racket Main, que será requerido para llamar el resto de implementaciones incluidas en un archivo rkt.

Esto será mediante la función REQUIRE Y PROVIDE.

Instrucciones de uso

Resultados y autoevaluación

Resultados esperados:
Se espera poder emular de forma consistente el conjunto de cartas y el juego directamente.

Conclusiones del trabajo

Tras realizar este trabajo, se puede concluir que el paradigma funcional utilizado es una herramienta bastante completa a la hora de trabajar. Al ser un paradigma diferente

Bibliografía

Gonzales, R. (2022). “Proyecto Semestral de Laboratorio”. Paradigmas de programación. Recuperado de:
2022_02 Laboratorio 1 - Documentos de Google