UNIVERSIDAD DE SANTIAGO DE CHILE
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA INFORMÁTICA

Informe De Laboratorio: Creación De Una Aplicación Estilo Photoshop En Swi-Prolog

Integrantes: Joaquín Saldivia
Profesor: Roberto Gonzales
Asignatura: Paradigmas de programación

UNIVERSIDAD DE SANTIAGO DE CHILE
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA INFORMÁTICA

Contenido

1.- Introducción

En este informe se expondrá una solución al problema de laboratorio el cual es realizar una aplicación para editar imágenes, utilizando el paradigma logico, donde se utilizará el lenguaje Swi-Prolog para la implementación de este.

1.1.- Descripción del problema

Se desea construir un programa que permita crear y editar imágenes al estilo de Photoshop o GIMP, agregando también la cualidad de profundidad a las imágenes. Se debe tener en cuenta que se tienen las siguientes estructuras como base: Imagen: Siendo la estructura principal del programa este se compone de un ancho, largo y pixeles, pudiendo aplicarles distintos tipos de edición. Pixeles: Son los que representan la imagen como tal teniendo una estructura de posición en el ancho y largo, color del píxel y su profundidad, este tendrá 3 tipos de formato y son pixbit, pixrgb y pixhex.

1.2.- Descripción del paradigma

El paradigma lógico es un paradigma declarativo, que funciona a base de hechos y reglas lógicas, posee una base de conocimiento la cual es el archivo con los predicados. Los predicados son lo que se quiere definir y se componen de hechos y reglas, estos se pueden escribir por ej: padre(juan, andres) donde tanto hechos como reglas comienzan en minúscula.
Los elementos mas importantes de este paradigma son:

Átomo: Son aquellas sobre lo que se basa el conocimiento que queremos explicar, deben ir en minúscula y empezar por una letra.

Predicado: Son lo que queremos explicar siendo como tal su “nombre” y los resultados y variables van en mayúscula.

Consulta: Son las preguntas que se hacen mediante la consola de Swi-Prolog y se busca en la base de conocimientos para retornar un true o false, también se puede dejan incógnitas (se escriben en mayuscula) para buscar valores que hacer verdadero al predicado

Hecho: Son aquellos que se declaran como bases de un predicado, se escriben de la misma forma que un predicado pero la información que estos proporcionan se toman como base una verdad.

1.3.- Objetivos

Como objetivo se tiene desarrollar el entendimiento del paradigma lógico através de Swi-Prolog con la idea de realizar por completo la solución al problema planteado

2.- Desarrollo

2.1.- Análisis del problema

Se tienen 2 entidades principales las que son la imagen y los pixeles que formaran parte de esta donde estos pueden ser de 3 tipos pixbit, pixhex y pixrgb, además la imagen tiene que aceptar distintos tipos de cambios también los pixeles deben aceptar cambios para operar con éxito los predicados en las imágenes y los predicados mencionadas que debe aceptar son:

imageIsPixmap, imageIsHexmap, imageIsBitmap, imageIsCompressed, imageFlipH, imageFlipV, imageCrop, imageRGBToHex, imageToHistogram, imageRotate90, imageCompress, imageInvertColorRGB, imageChangePixel, imageToString, imageDepthLayers, imageDecompress.

También se solicita que cada TDA utilizado este en un archivo propio.

2.2.- Diseño de solución

Al diseñar la solución se ocuparon los tipos de datos nativos del lenguaje y secrean 2 TDA que serán utilizados para simplificar el problema los que son:

TDA Image:

- Representación: una lista con la composición 2 int representando ancho y largo, una lista de pixeles y un color comprimido

- Constructor: se entregan solo los primeros 3 de los antes mencionados, esdecir: 2 int y una lista de pixeles que puede ser vacía para crear la lista imagen

- Predicados: Anexo 1

TDA Pixel:

- Representación: una lista que contiene 2 int indicando las posiciones x e y,un color que puede ser una lista de 3 int, un string o un int y la profundidad dependiendo del tipo de color que posea se le agregara un apellido al píxel.

- Constructor: al entregar 2 int un color y la profundidad se crea

una lista queserá conocido como pixbit, pixrgb o pixhex por lo antes mencionado.

- Predicados: No se hacen predicados adicionales

Luego la relación entre estos TDA se podrá ver en Anexo 2, además de mencionar que, dada la similitud de varios predicados estas se descomponen en predicados más pequeñas para lograr avances en más de una, utilizando principalmente recursión del tipo natural y de cola por la facilidad que otorgan al momento de recorrer una lista o crearla.

2.3.- Aspecto de implementación

El compilador utilizado es Swi-Prolog en versiones 8.4 o superior, se trabaja principalmente con listas, pero se hacen predicados para acceder a estas.

El código se estructura por TDA, para facilitar la edición a la hora de agregar contenido a alguno de estos, utilizando include para importar y exportar información de estos a otros archivos.

Para este laboratorio no se usan bibliotecas externas por lo que solo se defineque se utiliza el lenguaje Swi-Prolog.

2.4.- Ejemplos de uso

Se tiene un archivo main que contiene varios ejemplos y es requisito el tener los TDA Image y Pixel para su correcta ejecución.

Ahora para realizar operaciones fuera de estos ejemplos se debe tener en consideración que una imagen debe tener el mismo tipo de pixel y el anexo 3 se puede observar una manera de crear una imagen y la forma de aplicarle predicados.

También se espera poder aplicar a las imágenes los predicados de edición sin ningún problema, donde ante entradas erróneas no ocurran errores. Refiriéndose a los errores no se espera ningún error en concreto.

2.5.- Resultados y autoevaluación

Los resultados son los esperados, logrando hacer un programa completamente funcional, donde se intentó colocando entradas erróneas

aplicando más de una vez una función a una imagen e intentando imprimir imágenes comprimidas.

Se logran completar los 16 predicados propuestos para el proyecto

La autoevaluación radica entre valores de 0 (no implementado) hasta 1 (implementado y sin errores) aumentando en una escala de 0.25 y se puede observar en el anexo 4. Se les asigna a todos los predicados 1 dado que al momento de realizar diversas pruebas estas trabajan correctamente.

3.- Conclusión

Una vez terminado el trabajo se logra cumplir los objetivos, mejorando el entendimiento del paradigma lógico, así como también lograr una aplicación en Swi-Prolog para editar y crear imágenes.

La mayor limitación de este paradigma y lo que causo más dificultad al empezar este proyecto fue la forma en la que se implementa la recursión, dado que cada predicado pasaría a ser un condicional y fue dificultoso lograr conseguir las primeras recursiones sin recibir un false como respuesta al probar el predicado. Además no se notan diferencias entre paradigma lógico y funcional, sino más como un complemento donde el funcional es la función como tal y el lógico los condicionales que llevan a cada camino posible de la recursión en esta luego entre los lenguajes utilizados es la obtención de resultados, al desarrollar en Scheme si había un fallo se regresaba un error pero en Swi-Prolog se retorna false haciendo mas complicado el encontrar fallas.

4.- Bibliografía

1. Flores, V. (2022). “Proyecto semestral de laboratorio”. https://docs.google.com/document/d/14XEMPsuHicTIK7Hv3OuvW3wlG-SE1Ms9sJq9xwVLJSc/edit

5.- Anexos

1. Predicados del TDA Image:

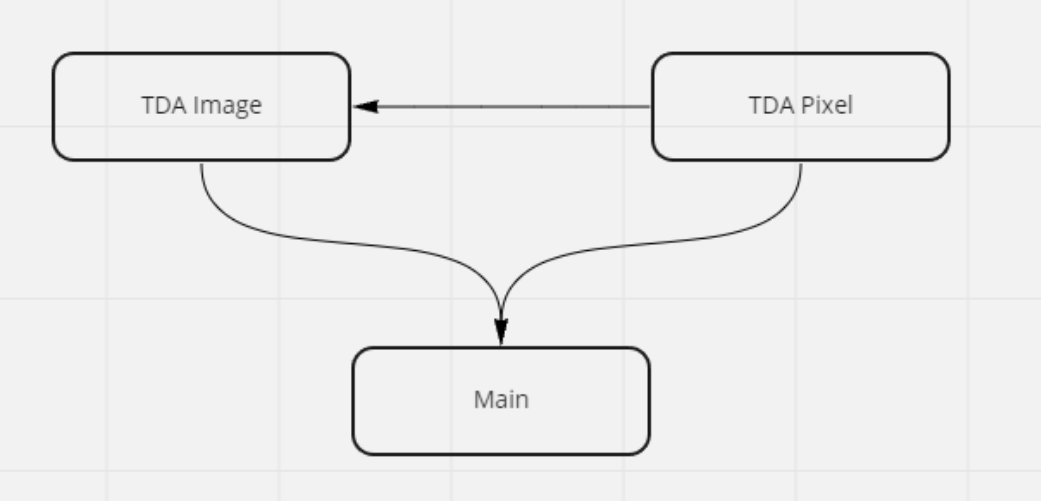
pertenencia	imageIsPixmap	verifica que la imagen tenga pixeles del tipo pixrgb-d
pertenencia	imageIsBitmap	verifica que la imagen tenga pixeles del tipo pixbit-d
pertenencia	imageIsHexmap	verifica que la imagen tenga pixeles del tipo pixhex-d
pertenencia	imagen	verifica que la imagen sea bit,hex o rgb
selector	getCompressedColor	si la imagen esta comprimida entrega el color eliminado
modificador	setCompressedColor	agrega a la imagen el color que fue eliminado
otros predicados	imageFlipH	voltea la imagen horizontalmente

otros predicados	imageFlipV	voltea la imagen verticalmente
otros predicados	imageCrop	recorta la imagen desde un (x0, y0) hasta un (x1, y1)
otros predicados	imageRGBToHex	transforma una imagen rgb a una imagen hexagesimal
otros predicados	imageToHistogram	entrega una lista de colores que aparecen en la imagen junto con la cantidad de veces que se repiten
otros predicados	imageRotate90	gira la imagen 90 grados en sentido horario
otros predicados	imageCompress	comprime la imagen eliminando el color más repetido
otros predicados	ImageInvertColorRGB	invierte el color rgb
otros predicados	imageToString	transforma una imagen en string
otros predicados	imageDepthLayers	crea una lista de imágenes, donde cada una de estas posee una profundidad diferente y los espacios vacios se rellenan con blanco
otros predicados	imageDecompress	descomprime una imagen perdiendo la información de profundidad
otros predicados	getMayor	entrega el color mas repetido en la lista retornada de histograma
otros predicados	add1	agrega 1 en el contador de veces que se repite en un color
otros predicados	myAppend	agrega un elemento al final de una lista

otros predicados	rellenarPixs	completa una lista de pixeles rellenando los faltantes con un color ingresado a una profundidad ingresada
otros predicados	flipPixsV	voltea los pixeles verticalmente
otros predicados	flipPixsH	voltea los pixeles horizontalmente
otros predicados	flipCrop	recorta los pixeles desde un (x0, y0) hasta un (x1, y1)
otros predicados	rgbToHexPixs	transforma los pixeles rgb a pixeles hexagesimal
otros predicados	genDepthogram	crea un lista de listas con pixeles separados por profundidad
otros predicados	genHistogram	crea un lista de listas con colores y la cantidad que se repiten
otros predicados	rotatePixs	gira los pixeles 90 grados en sentido horario
otros predicados	compressPixs	comprime los pixeles dado un color
otros predicados	inverColorPixsRGB	invierte el color rgb de una lista de pixeles
otros predicados	changePixs	cambia un pixel en especifico de una lista de pixeles
otros predicados	bitToString	convierte una lista de pixeles bit a string
otros predicados	rgbToString	convierte una lista de pixeles rgb a string
otros predicados	hexToString	convierte una lista de pixeles hex a string
otros predicados	addPix	agrega un pixel a una lista de pixeles
otros predicados	rellenarImagenesBit	rellena una imagen bit dado un color y profundidad
otros predicados	rellenarImagenesRGB	rellena una imagen rgb dado un color y profundidad

otros predicados	rellenarImagenesHex	rellena una imagen hex dado un color y profundidad

2. Relación entre TDA y main:



3. Ejemplos de uso:

```
?- pixrgb(0, 0, 1, 10, 30, 10, P1), pixrgb(0, 1, 0, 50, 132, 12, P2), pixrgb(1, 0, 0, 70, 122, 10, P3), pixrgb(1, 1, 1, 23, 133, 12, P4), image(2, 2, [P1, P2, P3, P4], I),imageCompress(
I,I2), imageDecompress(I2,I3).
P1 = [0, 0, [1, 10, 30], 10],
P2 = [0, 1, [0, 50, 132], 12],
P3 = [1, 0, [0, 70, 122], 10],
P4 = [1, 1, [1, 23, 133], 12],
I = [2, 2, [[0, 0, [1, 10|...], 10], [0, 1, [0|...], 12], [1, 0, [...|...]|...], [1, 1|...]], -1],
I2 = [2, 2, [[0, 1, [0, 50|...], 12], [1, 0, [0|...], 10], [1, 1, [...|...]|...]], [1, 10, 30]],
I3 = [2, 2, [[0, 0, [1, 10|...], 10], [1, 0, [0|...], 10], [0, 1, [...|...]|...], [1, 1|...]], -1] .


?- pixhex(0, 0, "#192341", 10, P1), pixhex(0, 1, "#00FFCC", 12, P2), pixhex(1, 0, "#00FFDD", 10, P3), pixhex(1, 1, "#77FFCC", 12, P4), image(2, 2, [P1, P2, P3, P4], I),imageDepthLayers(
I,ImageList).
P1 = [0, 0, "#192341", 10],
P2 = [0, 1, "#00FFCC", 12],
P3 = [1, 0, "#00FFDD", 10],
P4 = [1, 1, "#77FFCC", 12],
I = [2, 2, [[0, 0, "#192341", 10], [0, 1, "#00FFCC", 12], [1, 0, "#00FFDD"|...], [1, 1|...]], -1],
ImageList = [[2, 2, [[0, 0, "#FFFFFF", 12], [1, 0, "#FFFFFF"|...], [0, 1|...], [1|...]], -1], [2, 2, [[0, 0, "#192341"|...], [1, 0|...], [0|...], [...|...]], -1]] .

?- pixbit(0, 0, 1, 10, P1), pixbit(0, 1, 0, 12, P2), pixbit(1, 0, 0, 10, P3), pixbit(1, 1, 1, 12, P4), image(2, 2, [P1, P2, P3, P4], I),pixbit(1, 0, 1, 10, P3Mod),imageChangePixel(I,P3M
od,I2).
P1 = [0, 0, 1, 10],
P2 = [0, 1, 0, 12],
P3 = [1, 0, 0, 10],
P4 = [1, 1, 1, 12],
I = [2, 2, [[0, 0, 1, 10], [0, 1, 0, 12], [1, 0, 0|...], [1, 1|...]], -1],
P3Mod = [1, 0, 1, 10],
I2 = [2, 2, [[0, 0, 1, 10], [0, 1, 0, 12], [1, 0, 1|...], [1, 1|...]], -1] .

?- pixrgb(0, 0, 1, 10, 30, 10, P1), pixrgb(0, 1, 0, 50, 132, 12, P2), pixrgb(1, 0, 0, 70, 122, 10, P3), pixrgb(1, 1, 1, 23, 133, 12, P4), image(2, 2, [P1, P2, P3, P4], I),imageInvertCol
orRGB(I,I2).
P1 = [0, 0, [1, 10, 30], 10],
P2 = [0, 1, [0, 50, 132], 12],
P3 = [1, 0, [0, 70, 122], 10],
P4 = [1, 1, [1, 23, 133], 12],
I = [2, 2, [[0, 0, [1, 10|...], 10], [0, 1, [0|...], 12], [1, 0, [...|...]|...], [1, 1|...]], -1],
I2 = [2, 2, [[0, 0, [254, 245|...], 10], [0, 1, [255|...], 12], [1, 0, [...|...]|...], [1, 1|...]], -1] .
```

4. Autoevaluación:

Autoevaluacion	Puntaje
TDA	1
image	1
imageIsPixmap	1
imageIsHexmap	1
imageIsBitmap	1

imageIsCompressed	1
imageFlipH	1
imageFlipV	1
imageCrop	1
imageRGBToHex	1
imageToHistogram	1
imageRotate90	1
imageCompress	1
imageChangePixel	1
imageInvertColorRGB	1
imageToString	1
imageDepthLayers	1
imageDecompress	1