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

INFORME LABORATORIO 2: CREACIÓN DE SOFTWARE ESTILO *PHOTOSHOP* EN PROLOG

Nombre: Valentina Paz Campos Olguín
Profesor: Roberto González Ibáñez
Asignatura: Paradigmas de programación

26 de octubre de 2022

ÍNDICE

Introducción

Los paradigmas, en la vida cotidiana, significan un cambio en la forma de pensar, hacer o decir las cosas, haciendo que las mentes se abran debido al cambio drástico que conlleva el observar la misma situación con otros ojos.

Bajo este contexto, este informe se refiere al proyecto del segundo laboratorio del ramo de Paradigmas de programación. Se trabajará con la programación lógica a través del uso de SWI-Prolog versión 8.5.20.

El informe se dividirá en las secciones de descripción breve del problema, descripción breve del paradigma y conceptos asociados, análisis del problema, diseño de la solución, aspectos de implementación, instrucciones de uso, resultados, autoevaluación y a modo de término se realizará una conclusión con respecto al desarrollo del trabajo.

Descripción breve del problema

Se desea desarrollar un software de edición de imágenes estilo GIMP y Adobe Photoshop, pero desde una visión simplificada. Esta plataforma permite crear imágenes, determinar el tipo de imagen o si está comprimida, dar vuelta una imagen de forma horizontal y vertical, rotar una imagen, recortar una imagen, crear un histograma y más predicados.

Para desarrollar correctamente esta plataforma se debe tener en cuenta cada uno de los elementos principales que lo componen, por ejemplo:

Imágenes: Se deben crear a partir de un ancho, alto y una lista de píxeles que pueden ser de tipo pixbit, pixrgb o pixhex (de forma homogénea, no se pueden intercalar).

Píxeles: Cada píxel debe tener especificada su posición en x, en y, su color (ya sea en bit, RGB o hexadecimal) y su profundidad.

Predicados: Permiten construir, obtener y modificar una imagen, además de incluir operaciones que determinen la existencia de algún elemento de la imagen y finalmente incluir otros tipos de predicados que no entren en ninguna de las categorías descritas anteriormente.

Además, se debe tener en consideración que todo será implementado en SWI-Prolog en la versión 8.5.20.

Descripción del paradigma

En el paradigma lógico se describe una base de conocimientos o base de datos con hechos y reglas, donde se pueden realizar consultas sobre esta. Esto último significa que se le pueden realizar preguntas a la base de datos que se cargó previamente (Flores, 2020).

Hechos: Tipo de cláusula que define una relación entre términos. Son verdades asumidas por el intérprete de Prolog. Modela el problema a resolver en primera instancia relacionando términos entre sí (Flores, 2020).

Reglas: Tipo de cláusula que define una relación entre términos. Dependen de la conjunción de objetivos (Flores, 2020).

Está basado en tres mecanismos: **unificación**, **backtracking automático** y estructuras de datos basados en **árboles**.

Sustitución: Acción de dar un valor a una variable hasta hacer que la respuesta a la consulta sea verdad.
Unificación: Mecanismo utilizado para resolver una meta.
Backtracking: En términos simples, da un paso atrás mientras está resolviendo un problema y prueba de otra forma. Cuando se realiza este proceso se crea un árbol de las posibles soluciones que puede tomar. Si no encuentra al menos una solución, la respuesta a la consulta es falsa (Flores, 2020).

Análisis del problema

Para poder analizar el problema se debe subdividir este mismo, y para ello se examinará primeramente los tipos de datos abstractos y sus representaciones correspondientes:
pixbit = null | lista. Lista que contiene 4 enteros (int X int X int [0|1] X int).
pixrgb = null | lista. Lista que contiene 5 enteros (int X int X int [0 … 255] X int [0 … 255] X int [0 … 255] X int).
pixhex = null | lista. Lista que contiene 3 enteros y un string (int X int X string X int).
pixeles = null | lista. Lista que contiene un píxel y el resto es una lista (pixel X lista).
image = null | lista. Lista que contiene 2 enteros y una lista de píxeles (int X int X pixeles [pixbit* | pixrgb* | pixhex*]).
histogram = null | lista. Lista compuesta por una sublista y el resto de la lista, mientras que la sublista está compuesta por un entero y un color (int X color [Bit | R G B | Hex])

A continuación se procede a analizar los predicados principales que fueron implementadas:
imageIsBitmap: Predicado que determina si una imagen solo tiene píxeles de tipo pixbit.
imageIsPixmap: Predicado que determina si una imagen solo tiene píxeles de tipo pixrgb.
imageIsHexmap: Predicado que determina si una imagen solo tiene píxeles de tipo pixhex.
imageIsCompressed: Predicado que determina si una imagen está comprimida o no.
imageFlipH: Predicado que permite dar vuelta una imagen de forma horizontal.
imageFlipV: Predicado que permite dar vuelta una imagen de forma vertical.
imageCrop: Predicado que permite recortar una imagen a partir de un cuadrante.
imageRGBToHex: Predicado que permite convertir una imagen de tipo pixmap a hexmap.
imageToHistogram: Predicado que devuelve un histograma de frecuencias a partir de los colores en cada una de las imágenes.
imageRotate90: Predicado que permite rotar una imagen 90° a la derecha.
imageCompress: Predicado que permite comprimir una imagen eliminando los píxeles con mayor repetición de colores.
imageChangePixel: Predicado que permite reemplazar un píxel de una imagen.
imageInvertColorRGB: Predicado que permite invertir el color de un pixrgb-d.
imageToString: Predicado que transforma una imagen a representación string.
imageDepthLayers: Predicado que permite separar una imagen en capas en base a la profundidad en que se sitúan los píxeles. El resultado es una lista de imágenes donde cada una agrupa los píxeles que se sitúan en el mismo nivel de profundidad. Si un píxel no tiene la misma profundidad que otro, se sustituye su color por blanco.

Diseño de la solución

Para darle forma a la solución hay que tomar en cuenta que se deben crear tipos de datos para poder realizar las operaciones principales. En este caso, se escogerán listas para la implementación de lo anterior descrito. Estas operaciones tienen una representación y se subdividen según su tarea a cumplir, ya sea como constructor, selector, modificador, predicado de pertenencia y otros predicados que no califiquen como ninguna de las anteriores.

TDA pixbit

Representación: Una lista donde el primer elemento es su posición en x, el siguiente la posición en y, el siguiente el valor del bit y el último valor es la profundidad.

Constructor: Se requiere de 4 enteros y un pixbit.

Selectores, modificadores, predicados de pertenencia y otros predicados estarán en anexos.

TDA pixrgb

Representación: Una lista donde el primer elemento es su posición en x, el siguiente la posición en y, el siguiente el valor del canal Red, el siguiente el valor del canal Green, el siguiente el valor del canal Blue y el último valor es la profundidad.

Constructor: Se requiere de 6 enteros y un pixrgb.

Selectores, modificadores, predicados de pertenencia y otros predicados estarán en anexos.

TDA pixhex

Representación: Una lista donde el primer elemento es su posición en x, el siguiente la posición en y, el siguiente valor su color en hexadecimal y el último valor es la profundidad.

Constructor: Se requiere de 3 enteros, 1 string y un pixhex.

Selectores, modificadores, predicados de pertenencia y otros predicados estarán en anexos.

TDA image

Representación: Una lista donde el primer elemento sea el ancho de la imagen, el segundo elemento sea el alto, y el tercer elemento es una lista de píxeles que pueden ser de tipo pixbit, pixrgb o pixhex (homogéneas, es decir, no se pueden intercalar).

Luego, se crean los predicados selectores donde se pueden obtener los valores de las coordenadas, los canales de color, la profundidad, la lista de pixeles, los bits de una lista de píxeles tipo pixbit y más.

Se crean los predicados modificadores, los cuales cambian los valores de las posiciones, cambiar el largo, ancho, eliminar píxeles, cambiar los colores de los píxeles y permiten convertir una imagen a una representación string, entre otras.

Se crean además los predicados de pertenencia, donde se evalúa si la imagen es de tipo bitmap, pixmap o hexmap, y si está comprimida.

La gran mayoría de estas operaciones incluyeron recursividad de tipo cola y natural, debido a que se necesitó analizar cada elemento de los TDA's y además se necesitó crear predicados extra que no pertenecen a ningún TDA específico para llegar a los resultados deseados.

Para lograr unir todo lo anterior bastó con tener un archivo llamado main_21305689_CamposOlguin.pl donde están todos los requerimientos funcionales, mientras que en tdapixbit_21305689_CamposOlguin.pl , tdapixrgb_21305689_CamposOlguin.pl y tdapixhex_21305689_CamposOlguin.pl los TDA's.

Aspectos de implementación

El proyecto deberá tener la siguiente estructura:
Primero están las especificaciones de los TDA, luego cada predicado está separado según lo que realiza con la imagen, siendo constructores, selectores, modificadores, predicados de pertenencia y otros predicados. En cada una de estas se detalla su dominio y la descripción del predicado.
Se decidió hacer de esta forma para un mayor orden y limpieza en el código, de manera que sea lo más legible posible.
No se empleó ninguna biblioteca externa y se usó SWI-Prolog versión 8.5.20.

Instrucciones de uso

Para crear una imagen se debe ocupar el predicado image, ingresando los parámetros que fueron descritos anteriormente.
Luego, se pueden ejecutar los predicados pertinentes a la imagen, ya sea imageFlipH, imageFlipV, imageToHistogram, imageDepthLayers, imageRotate90 o cualquiera de los otros implementados.
Finalmente se mostrará por consola el primer resultado que se adecúa a la consulta, devolviendo el valor que debe tener la variable pedida para que la respuesta a la consulta sea verdadera.
Un ejemplo de esto es:
pixbit(0, 0, 1, 0, P1), pixbit(0, 1, 0, 10, P2), image(1, 2, [P1, P2], I).
El cual devuelve, además de los valores de P1 y P2:
I = [1, 2, [[0, 0, 1, 0], [0, 1, 0, 10]]]
Se espera que los resultados mostrados por consola tengan concordancia con la estructura de la imagen, respetando las posiciones en x, y y demás valores ya sea a la hora de crear la imagen como al momento de consultar si pertenece a una imagen tipo bitmap o más.
Podrían existir errores como definir una imagen de 0 x 0, pero se asume que quien emplee el programa sabe que no puede existir una imagen sin dimensiones.
En la zona de anexos (desde la tabla N°1 hasta la N°4) se encuentran los distintos ejemplos de uso.

Resultados y autoevaluación

Se realizaron más de 72 pruebas para alcanzar estados críticos de ejecución, ya sea a través de una lista de píxeles nula como dividir una imagen en diferentes listas según su profundidad para poder ordenarlas, cambiando profundidades, incrementando valores, disminuyendo valores y más.
Se realizaron 17 de los 18 predicados descritos en el documento, sin completarse el predicado decompress debido a que requería un cambio en el TDA image para que pudiera incluir los píxeles que han sido comprimidos. De estos 17 predicados el 100% funcionó correctamente.
Los resultados obtenidos fueron los esperados debido a que el programa logró adecuarse a cada una de las situaciones extremas a las que se le puso a prueba y las pasó sin problemas.

Conclusiones

En el software no existieron limitaciones relevantes a la hora de desarrollar el proyecto, a excepción de saber cómo empezar a darle forma a este y que tuviera en consideración que debe funcionar la gran mayoría de veces. Sin embargo, una vez ya sabiendo cómo empezar todo se puede hacer con mayor fluidez.

Algunas dificultades fueron el cambio de paradigma, ya que la costumbre de trabajar con el paradigma funcional y el paradigma orientado a objetos aún está presente.

Finalmente, se puede concluir que las representaciones e implementaciones fueron las adecuadas para abordar el problema, ya que el proyecto cumplió con todas las expectativas de funcionamiento.

Bibliografía

01 - INTRODUCCIÓN AL PARADIGMA LÓGICO. (Flores V., 2020). [Vídeo]. uvirtual.usach.cl. Recuperado el 26 de octubre 2022, de https://uvirtual.usach.cl/moodle/mod/hvp/view.php?id=156617

03 - PARTES DE UN PROGRAMA LÓGICO EN PROLOG. (Flores V., 2020). [Vídeo]. uvirtual.usach.cl. Recuperado el 26 de octubre 2022, de https://uvirtual.usach.cl/moodle/mod/hvp/view.php?id=156622

04 - HECHOS EN PROLOG. (Flores V., 2020). [Vídeo]. uvirtual.usach.cl. Recuperado el 26 de octubre 2022, de https://uvirtual.usach.cl/moodle/mod/hvp/view.php?id=156623

05 - REGLAS EN PROLOG. (Flores V., 2020). [Vídeo]. uvirtual.usach.cl. Recuperado el 26 de octubre 2022, de https://uvirtual.usach.cl/moodle/mod/hvp/view.php?id=156624

09 - UNIFICACIÓN Y BACKTRACKING. (Flores V., 2020). [Vídeo]. uvirtual.usach.cl. Recuperado el 26 de octubre 2022, de https://uvirtual.usach.cl/moodle/mod/hvp/view.php?id=156629

Anexos

Tabla N°1: Predicados del programa principal

Nombre de predicado	Tipo de predicado	Ejemplo de uso	Resultado
image	Constructor	pixbit(0, 0, 1, 0, P1), pixbit(0, 1, 0, 10, P2), image(1, 2, [P1, P2], I).	I = [1, 2, [[0, 0, 1, 0], [0, 1, 0, 10]]], P1 = [0, 0, 1, 0], P2 = [0, 1, 0, 10]
imageToHistogram	Constructor	pixbit(0, 0, 1, 0, P1), pixbit(0, 1, 0, 10, P2), image(1, 2, [P1, P2], I), imageToHistogram(I, His).	His = [[1, 1], [0, 1]], I = [1, 2, [[0, 0, 1, 0], [0, 1, 0, 10]]], P1 = [0, 0, 1, 0], P2 = [0, 1, 0, 10]
imageDepthLayers	Constructor	pixbit(0, 0, 1, 0, P1), pixbit(0, 1, 0, 10, P2), image(1, 2, [P1, P2], I), imageDepthLayers(I, Lista).	Lista = [[1, 2, [[0, 0, 1, 0], [0, 1, 1, 0]]], [1, 2, [[0, 0, 1, 10], [0, 1, 0, 10]]]], I = [1, 2, [[0, 0, 1, 0], [0, 1, 0, 10]]], P1 = [0, 0, 1, 0], P2 = [0, 1, 0, 10]
imageFlipH	Modificador	pixbit(0, 0, 1, 0, P1), pixbit(0, 1, 0, 10, P2), image(1, 2, [P1, P2], I), imageFlipH(I, I2).	I = I2, I2 = [1, 2, [[0, 0, 1, 0], [0, 1, 0, 10]]], P1 = [0, 0, 1, 0], P2 = [0, 1, 0, 10]
imageFlipV	Modificador	pixbit(0, 0, 1, 0, P1), pixbit(0, 1, 0, 10, P2), image(1, 2, [P1, P2], I), imageFlipV(I, I2).	I = [1, 2, [[0, 0, 1, 0], [0, 1, 0, 10]]], I2 = [1, 2, [[0, 0, 0, 10], [0, 1, 1, 10]]], P1 = [0, 0, 1, 0], P2 = [0, 1, 0, 10]
imageCrop	Modificador	pixbit(0, 0, 1, 0, P1), pixbit(0, 1, 0, 10, P2), image(1, 2, [P1, P2], I), imageCrop(I, 0, 0, 0, 0, I2).	I = [1, 2, [[0, 0, 1, 0], [0, 1, 0, 10]]], I2 = [1, 1, [[0, 0, 0, 10]]], P1 = [0, 0, 1, 0], P2 = [0, 1, 0, 10]
imageRGBToHex	Modificador	pixrgb(0, 0, 255, 255, 255, 10, P1), pixrgb(1, 0, 0, 0, 0, 20, P2), pixrgb(0, 1, 255, 0, 255, 30, P3), pixrgb(1, 1, 0, 255, 0, 40, P4), image(2, 2, [P1, P2, P3, P4], I), imageRGBToHex(I, I2).	I = [2, 2, [[0, 0, 255, 255, 255, 10], [1, 0, 0, 0, 0, 20], [0, 1, 255, 0, 255, 30], [1, 1, 0, 255, 0, 40]]], I = [2, 2, [[0, 0, 255, 255, 255, 10], [1, 0, 0, 0, 0, 20], [0, 1, 255, 0, 255, 30], [1, 1, 0, 255, 0, 40]]], P1 = [0, 0, 255, 255, 255, 10], P2 = [1, 0, 0, 0, 0, 20], P3 = [0, 1, 255, 0, 255, 30], P4 = [0, 1, 255, 0, 255, 30]
imageRotate90	Modificador	pixbit(0, 0, 1, 0, P1), pixbit(0, 1, 0, 10, P2), image(1, 2, [P1, P2], I), imageRotate90(I, I2).	I = [1, 2, [[0, 0, 1, 0], [0, 1, 0, 10]]], I2 = [2, 1, [[0, 0, 0, 10], [1, 0, 1, 0]]], P1 = [0, 0, 1, 0], P2 = [0, 1, 0, 10]

imageCompress	Modificador	pixbit(0, 0, 1, 0, P1), pixbit(0, 1, 0, 10, P2), image(1, 2, [P1, P2], I), imageCompress(I, I2).	I = [1, 2, [[0, 0, 1, 0], [0, 1, 0, 10]]], I2 = [1, 2, [[0, 0, 1, 0]]] P1 = [0, 0, 1, 0], P2 = [0, 1, 0, 10]
imageChangePixel	Modificador	pixbit(0, 0, 1, 0, P1), pixbit(0, 1, 0, 10, P2), image(1, 2, [P1, P2], I), pixbit(0, 0, 0, 20, P1_mod),imageChangePixel(I, P1_mod, I2).	I = [1, 2, [[0, 0, 1, 0], [0, 1, 0, 10]]], I2 = [1, 2, [[0, 0, 1, 20], [0, 1, 0, 10]]] P1 = [0, 0, 1, 0], P1_mod = [0, 0, 0, 20], P2 = [0, 1, 0, 10]
imageInvertColorRGB	Modificador	pixrgb(0, 0, 255, 255, 255, 10, P1), imageInvertColorRGB(P1, P2).	P1 = [0, 0, 255, 255, 255, 10], P2 = [0, 0, 0, 0, 0, 10]
imageToString	Modificador	pixbit(0, 0, 1, 0, P1), pixbit(0, 1, 0, 10, P2), image(1, 2, [P1, P2], I), imageToString(I, Str).	I = [1, 2, [[0, 0, 1, 0], [0, 1, 0, 10]]], P1 = [0, 0, 1, 0], P2 = [0, 1, 0, 10], Str = “1\t\n0\t\n\n”
imageIsBitmap	De pertenencia	pixbit(0, 0, 1, 0, P1), pixbit(0, 1, 0, 10, P2), image(1, 2, [P1, P2], I), imageIsBitmap(I).	I = [1, 2, [[0, 0, 1, 0], [0, 1, 0, 10]]], P1 = [0, 0, 1, 0], P2 = [0, 1, 0, 10]
imageIsPixmap	De pertenencia	pixbit(0, 0, 1, 0, P1), pixbit(0, 1, 0, 10, P2), image(1, 2, [P1, P2], I), imageIsPixmap(I).	false
imageIsHexmap	De pertenencia	pixbit(0, 0, 1, 0, P1), pixbit(0, 1, 0, 10, P2), image(1, 2, [P1, P2], I), imageIsHexmap(I).	false
imageIsCompressed	De pertenencia	pixbit(0, 0, 1, 0, P1), pixbit(0, 1, 0, 10, P2), image(1, 2, [P1, P2], I), imageIsCompressed(I).	false

Tabla N°2: Predicados de TDA pixbit

Nombre de predicado	**Tipo de predicado**	**Ejemplo de uso**	**Resultado**
pixbit	Constructor	pixbit(0, 0, 1, 0, P1).	P1 = [0, 0, 1, 0]
getx	Selector	pixbit(0, 0, 1, 0, P1), getx(P1, X).	X = 0
gety	Selector	pixbit(0, 0, 1, 0, P1), gety(P1, Y).	Y = 0
getbit	Selector	pixbit(0, 0, 1, 0, P1), getbit(P1, B).	B = 1
getd	Selector	pixbit(0, 0, 1, 0, P1), gerd(P1, D).	D = 0
isPixbit	De pertenencia	pixbit(0, 0, 1, 0, P1), isPixbit(P1).	P1 = [0, 0, 1, 0]

Tabla N°3: Predicados de TDA pixrgb

Nombre de predicado	Tipo de predicado	Ejemplo de uso	Resultado
pixrgb	Constructor	pixrgb(0, 1, 34, 50, 255, 80, P1).	P1 = [0, 1, 34, 50, 255, 80]
getx	Selector	pixrgb(0, 1, 34, 50, 255, 80, P1), getx(P1, X).	X = 0
gety	Selector	pixrgb(0, 1, 34, 50, 255, 80, P1), gety(P1, Y).	Y = 1
getr	Selector	pixrgb(0, 1, 34, 50, 255, 80, P1), getr(P1, R).	R = 34
getg	Selector	pixrgb(0, 1, 34, 50, 255, 80, P1), getg(P1, G).	G = 50
getb	Selector	pixrgb(0, 1, 34, 50, 255, 80, P1), getb(P1, B).	B = 255
getd_rgb	Selector	pixrgb(0, 1, 34, 50, 255, 80, P1), getd_rgb(P1, D).	D = 80
isPixrgb	De pertenencia	pixrgb(0, 1, 34, 50, 255, 80, P1), isPixrgb(P1).	P1 = [0, 1, 34, 50, 255, 80]

Tabla N°4: Predicados de TDA pixhex

Nombre de predicado	Tipo de predicado	Ejemplo de uso	Resultado
pixhex	Constructor	pixhex(1, 1, “0000FF”, 5, P1).	P1 = [1, 1, “0000FF”, 5]
getx	Selector	pixhex(1, 1, “0000FF”, 5, P1), getx(P1, X).	X = 1
gety	Selector	pixhex(1, 1, “0000FF”, 5, P1), gety(P1, Y).	Y = 1
gethex	Selector	pixhex(1, 1, “0000FF”, 5, P1), gethex(P1, Hex).	Hex = “0000FF”
getd	Selector	pixhex(1, 1, “0000FF”, 5, P1), getd(P1, D).	D = 5
isPixhex	De pertenencia	pixhex(1, 1, “0000FF”, 5, P1), isPixhex(P1).	P1 = [1, 1, “0000FF”, 5]