UNIVERSIDAD DE GUADALAJARA

CENTRO UNIVERSITARIO DE CIENCIAS EXACTAS E INGENIERIAS

SEMINARIO DE SOLUCIÓN A PROBLEMAS DE ALGORITMIA

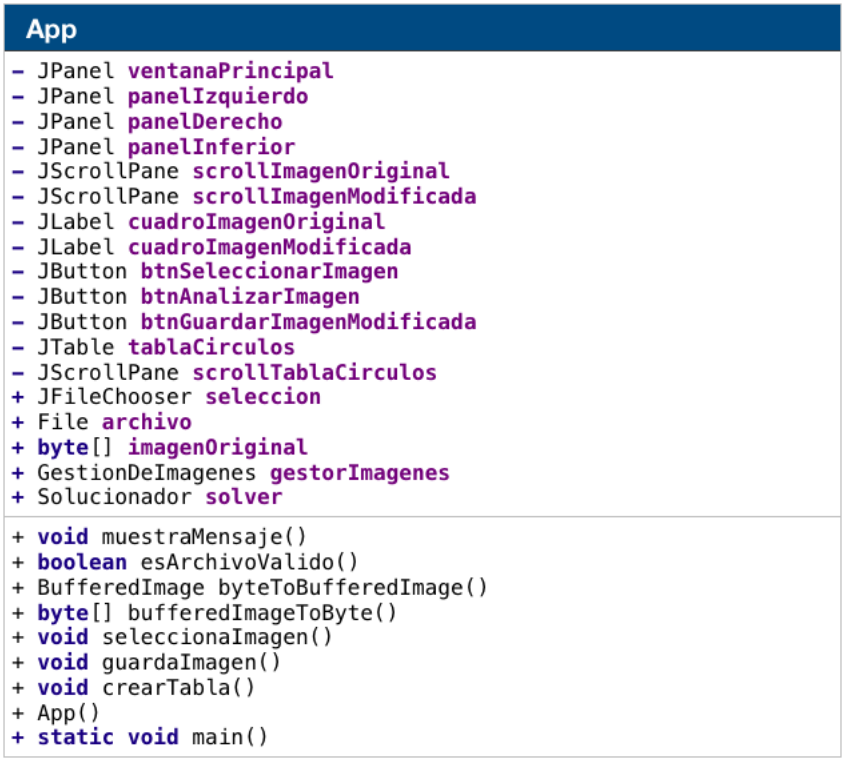
Actividad 1: Localización de círculos

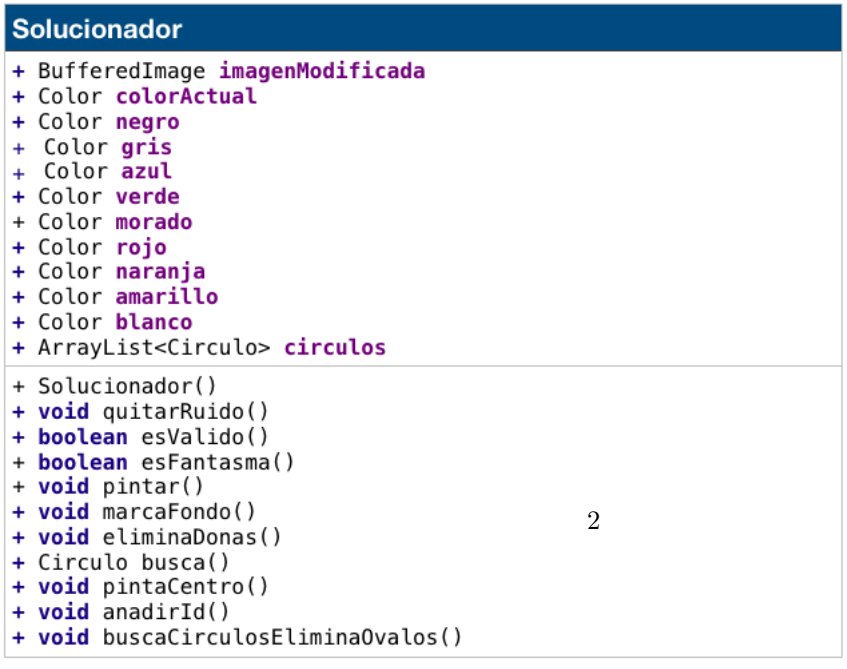
Presentado a:
David Alejandro Gómez Anaya
Departamento de Ciencias Computacionales

Presentado por:
Abraham Baltazar Murillo Sandoval, 218744368
Seccion D15, 2019 B

Contents

1 Diagrama de clases propuesto





Punto

```
+ int x
+ int y
+ Punto()
```

Circulo

```
+ Punto inicial
+ Punto arriba, abajo
+ Punto izquierda, derecha
+ Punto centroide
+ boolean tocaBorde
+ Circulo()
+ boolean esCirculo()
+ void calculaCentroide()
+ void expande()
+ int radioHorizontal()
+ int radioVertical()
+ int radio()
```

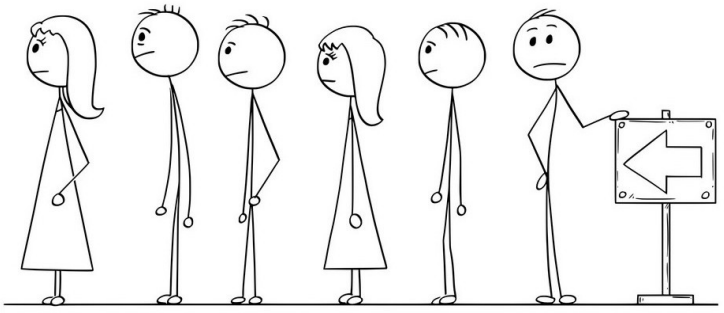
2 Objetivo

Crear un sistema capaz de identificar círculos negros de óvalos negros en una base blanca, ignorando figuras pegadas al borde de la misma y donas (círculo con círculo interior). Mostrar únicamente en la imagen procesada los círculos válidos, etiquetados, además de una tabla con la posición de su centro y radio de cada círculo respectivo.

3 Marco teórico

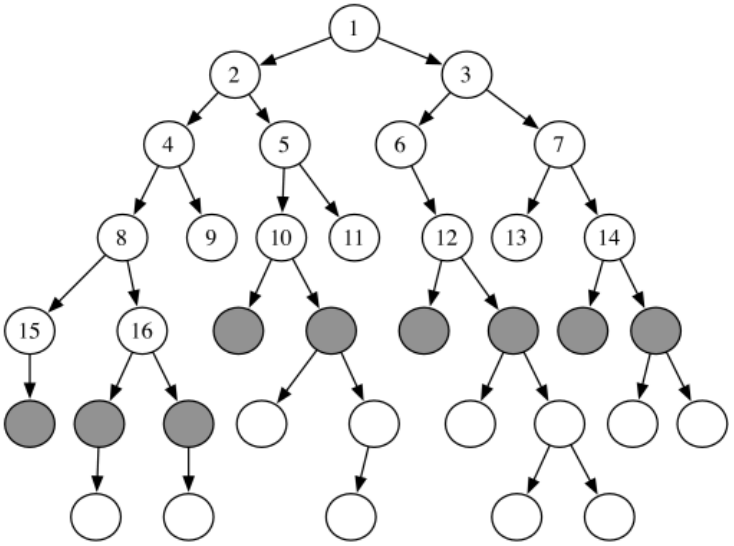
Cola (Queue):

Una cola es una estructura de datos que nos permite tener los datos de acuerdo al orden de entrada. Para un ejemplo más claro, ir a las tortillas, sólo se pueden remover elementos de un lado de la misma generalmente para procesarlos (atenderlos) y agregar nuevos en el extremo contrario, para analizarlos cuando llegue su turno (los que llegan a formarse para comprar).



Algoritmo Breadth-first search (BFS):

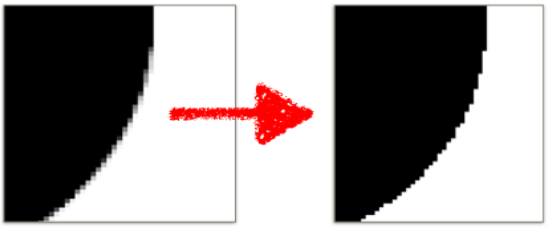
Es un algoritmo de teoría de grafos-árboles que nos permite desde uno o varios puntos de partida explorar los "vecinos" adyacentes a él, y generalmente encontrar la distancia mínima para alcanzarlo (puede extenderse a otras aplicaciones).



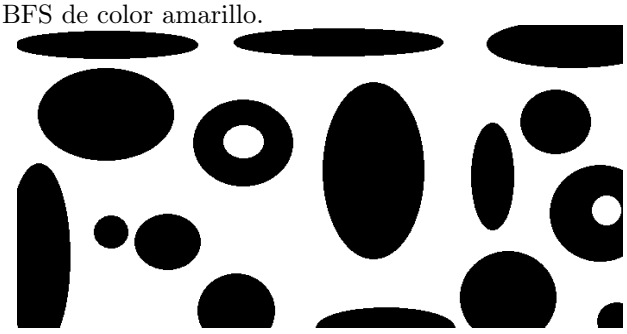
4 Desarrollo

Para esta actividad hice uso de mucho análisis, imaginación y múltiples BFS para colorear conjuntos de pixeles (figuras, fondos).

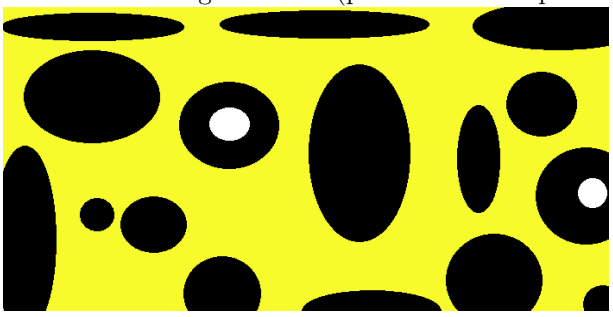
1. Todo pixel que *no* era blanco, ahora es negro, esto con el objetivo de eliminar el ruido en el borde de las imágenes.



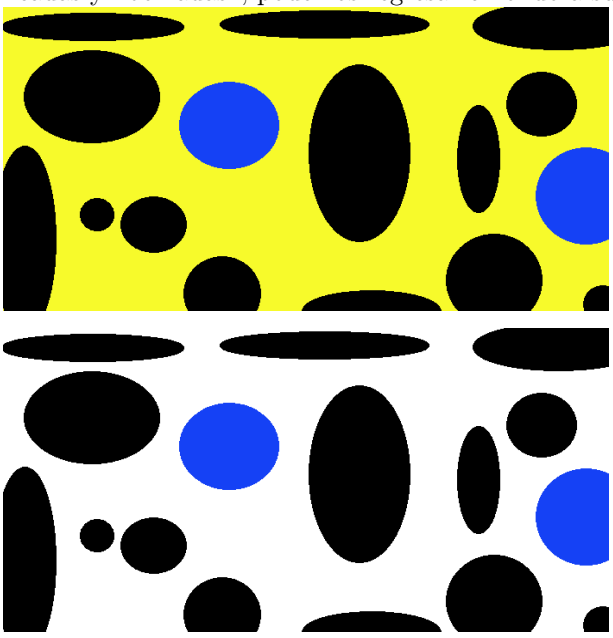
2. Agregué un borde imaginario a la imagen, es decir, el pixel(-1, -1) ahora es el topmost-leftmost que antes ocupada el pixel(0, 0), desde ahí todo el fondo está conectado por color blanco, entonces lo coloree mediante una BFS de color amarillo.



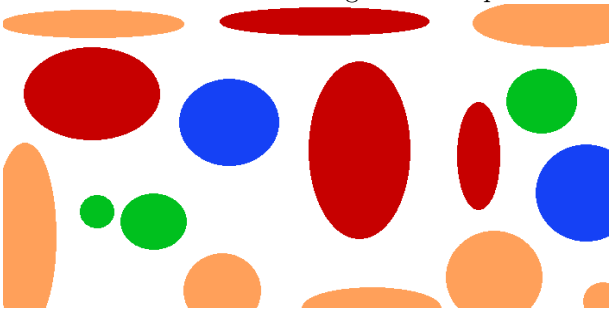
3. Una dona es una figura que en su interior tendrá blanco, y ahora como todo el fondo ahora es amarillo, podemos *garantizar* que lo es si cumple con lo dicho anteriormente; pintamos su interior de negro y posteriormente todo lo de negro de azul (para identificar que fue una dona).



4. Ya con las donas identificadas y "borradas", podemos regresar el fondo a su color original.



5. Todo pixel negro, es un candidato a ser una de las siguientes 3 opciones:

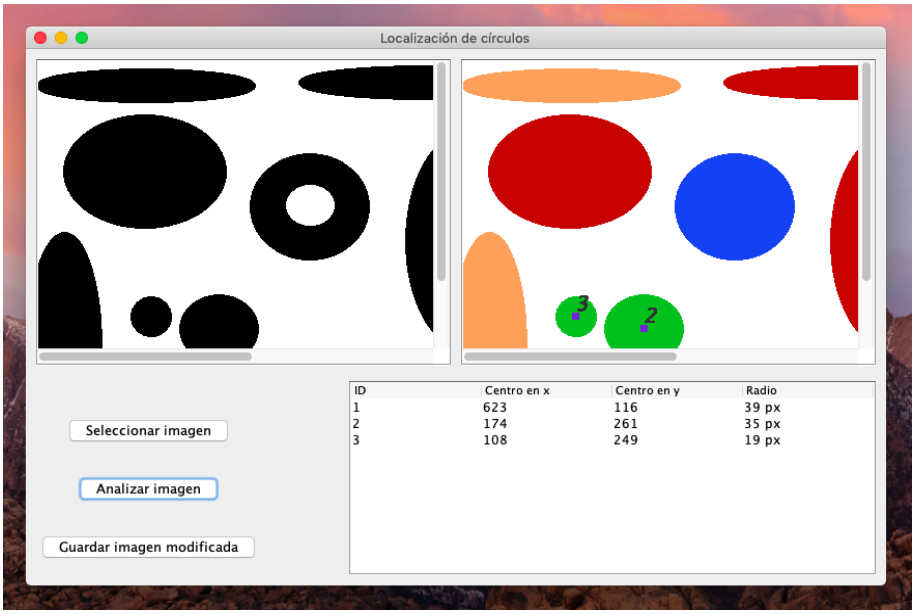


   (a) Círculo (verde), se puede saber que lo es si no toca ningún borde, entonces tiene 4 puntos importantes "arriba", "abajo", "izquierda", "derecha"", con los que podemos calcular el centro y por lo tanto el `radio()` del círculo en cuestión.
   Además debe de cumplir que la diferencia entre el `radioVertical()` y `radioHorizontal()` sea menor a 10 pixeles.

(b) Óvalo (rojo), es un "falso circulo", es decir, cumple con todas las características de un círculo excepto por la más importante, su diferencia es mayor que 10 pixeles.

(c) Figura al borde (naranja), esta es cuando toca alguno de los bordes, fácil de calcular mientras se realiza la BFS e intenta ir a una posición inválida (fuera de rango).

6. Ya que tenemos pintado todo, agregamos un cuadro de 7 x 7 pixeles para marcar el centro y su número respectivo.
   De igual manera mostramos una tabla con el ID del círculo, su centro en x y y, y su radio.



5 Códigos utilizados

Se utilizó el siguiente código para quitar el ruido que tuviera la imagen.

```
%public void quitarRuido() {
    for (int y = 0; y < imagenModificada.getHeight(); y++) {
        for (int x = 0; x < imagenModificada.getWidth(); x++) {
            colorActual = new Color(imagenModificada.getRGB(x, y));
            if (colorActual.getRGB() != blanco.getRGB()) {
                imagenModificada.setRGB(x, y, negro.getRGB());
            }
        }
    }
}
```

Se utilizó el siguiente código para pintar desde el pixel(x, y) todos aquellos que fueran de color *from* a color *to*.

```
%public void pintar(int x, int y, Color from, Color to){
      Queue<Punto> qu = new LinkedList<>();
      qu.add(new Punto(x, y));
      imagenModificada.setRGB(x, y, to.getRGB());
      while (!qu.isEmpty()) {
          Punto actual = qu.poll();
          for (int k = 0; k < 4; k++) {
              int nx = actual.x + dx[k];
              int ny = actual.y + dy[k];
              if (!esValido(nx, ny)) {
                  continue;
              }
              colorActual = new Color(imagenModificada.getRGB(nx, ny));
              if (colorActual.getRGB() == from.getRGB() ) {
                  qu.add(new Punto(nx, ny));
                  imagenModificada.setRGB(nx, ny, to.getRGB());
              }
          }
      }
  }
```

Se utilizó el siguiente código para pintar el fondo de color *from* a color *to*.

```
%public void marcaFondo(Color from, Color to){
    boolean[][] vis = new boolean[imagenModificada.getWidth() +
       2][imagenModificada.getHeight() + 2];
    for(int y = 0; y < imagenModificada.getHeight(); y++){
        for(int x = 0; x < imagenModificada.getWidth(); x++){
            vis[x + 1][y + 1] = true;
        }
    }
    for(int y = 0; y < imagenModificada.getHeight(); y++){
        for(int x = 0; x < imagenModificada.getWidth(); x++){
            colorActual = new Color(imagenModificada.getRGB(x, y));
            if( colorActual.getRGB() == from.getRGB() ) {
                vis[x + 1][y + 1] = false;
            }
        }
    }
    Queue<Punto> qu = new LinkedList<>();
    qu.add(new Punto(-1, -1)); // Punto fantasma (-1, -1)
    vis[-1 + 1][-1 + 1] = true;
    while (!qu.isEmpty()) {
        Punto actual = qu.poll();
        for (int k = 0; k < 4; k++) {
            int nx = actual.x + dx[k];
            int ny = actual.y + dy[k];
            if( esValido(nx, ny) ){
                colorActual = new Color(imagenModificada.getRGB(nx, ny));
            }else if( esFantasma(nx, ny) ){
                colorActual = to;
            }else{
                continue;
            }
            if (  !vis[nx + 1][ny + 1] ) {
                qu.add(new Punto(nx, ny));
                if( esValido(nx, ny) ){
                    imagenModificada.setRGB(nx, ny, to.getRGB());
                }
                vis[nx + 1][ny + 1] = true;
            }
        }
    }
}
```

Se utilizó el siguiente código para eliminar las donas.

```
%public void eliminaDonas(){
    for(int y = 0; y < imagenModificada.getHeight(); y++){
        for(int x = 0; x < imagenModificada.getWidth(); x++){
            colorActual = new Color(imagenModificada.getRGB(x, y));
            if( colorActual.getRGB() == blanco.getRGB() ){
                pintar(x, y, blanco, negro);
                pintar(x, y, negro, azul);
            }
        }
    }
}
```

Se utilizó el siguiente código para buscar un círculo.

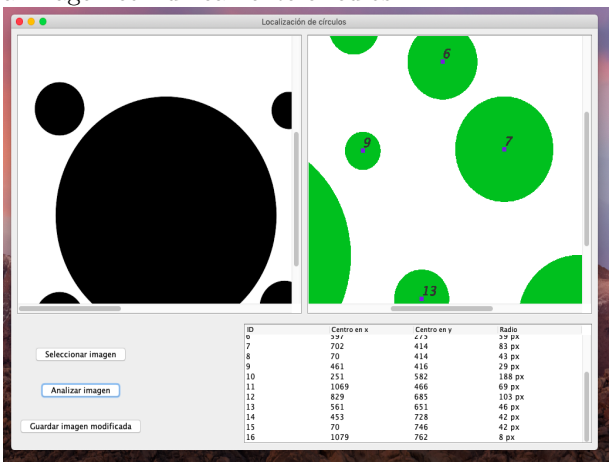
```
%public Circulo busca(int x, int y, Color from, Color to){
    Circulo cir = new Circulo(x, y); // Punto en (x, y)
    Queue<Punto> qu = new LinkedList<>();
    qu.add(new Punto(x, y));
    imagenModificada.setRGB(x, y, to.getRGB());
    while (!qu.isEmpty()) {
        Punto actual = qu.poll();
        for (int k = 0; k < 4; k++) {
            int nx = actual.x + dx[k];
            int ny = actual.y + dy[k];
            if (!esValido(nx, ny)) {
                cir.tocaBorde = true;
                continue;
            }
            colorActual = new Color(imagenModificada.getRGB(nx, ny));
            if (colorActual.getRGB() == from.getRGB() ) {
                qu.add(new Punto(nx, ny));
                cir.expande(nx, ny);
                imagenModificada.setRGB(nx, ny, to.getRGB());
            }
        }
    }
    cir.calculaCentroide();
    return cir;
}
```

Se utilizó el siguiente código para buscar círculos y eliminar figuras al borde u óvalos.

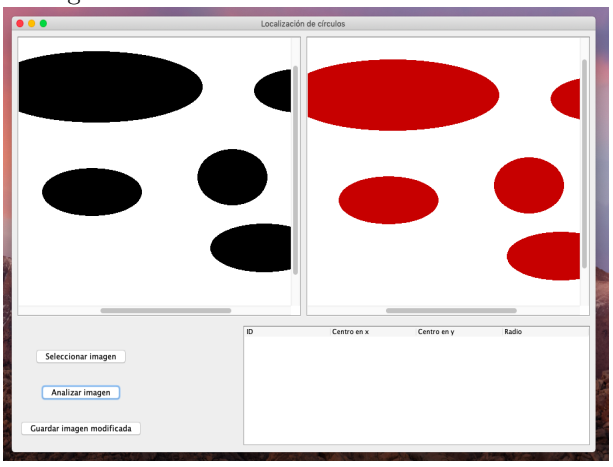
```
%public void buscaCirculosEliminaOvalos(){
    circulos = new ArrayList<>();
    Circulo cir;
    for(int y = 0; y < imagenModificada.getHeight(); y++) {
        for (int x = 0; x < imagenModificada.getWidth(); x++) {
            colorActual = new Color(imagenModificada.getRGB(x, y));
            if (colorActual.getRGB() == negro.getRGB()) {
                cir = busca(x, y, negro, verde);
                if( cir.tocaBorde ){
                    pintar(cir.inicial.x, cir.inicial.y, verde, naranja);
                }else if( cir.esCirculo() ){
                    pintaCentro(cir);
                    circulos.add(cir);
                    anadirId(cir, circulos.size());
                }else{
                    pintar(cir.inicial.x, cir.inicial.y, verde, rojo);
                }
            }
        }
    }
}
```

6 Pruebas y resultados

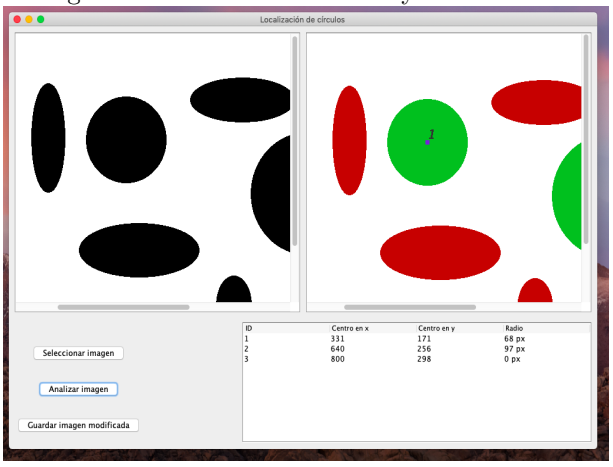
1. Prueba realizada en una imagen con únicamente círculos.



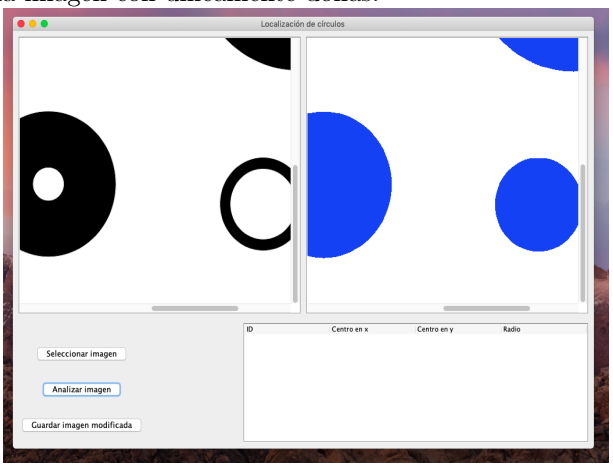
2. Prueba realizada en una imagen con únicamente óvalos.



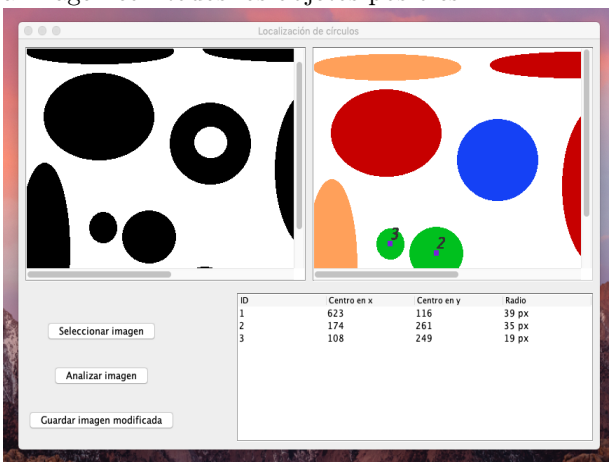
3. Prueba realizada en una imagen con únicamente círculos y óvalos.



4. Prueba realizada en una imagen con únicamente donas.



5. Prueba realizada en una imagen con todos los objetos posibles.



7 Conclusiones

En esta actividad se puso a prueba la imaginación y las buenas ideas para no caer en una solución demasiado costosa en tiempo, puesto que podía haber imágenes muy grandes, y por tanto con una complejidad muy alta podría hacer que nuestro ordenador llegará a trabarse.
Combinar estrategias de programación competitiva y recuerdos de la infancia (paint) hizo el proceso para resolverlo más sencillo y divertido. Una complejidad donde se procesa cada pixel exactamente una vez es la mejor en todos los aspectos, en este caso recurrir a una BFS para pintar e identificar lo que se está procesando.

8 Apéndice(s)

https://docs.oracle.com/javase/7/docs/api/javax/swing/package-summary.html (Java Swing)
https://www.youtube.com/watch?v=KiCBXu4P-2Y (BFS en un tablero)