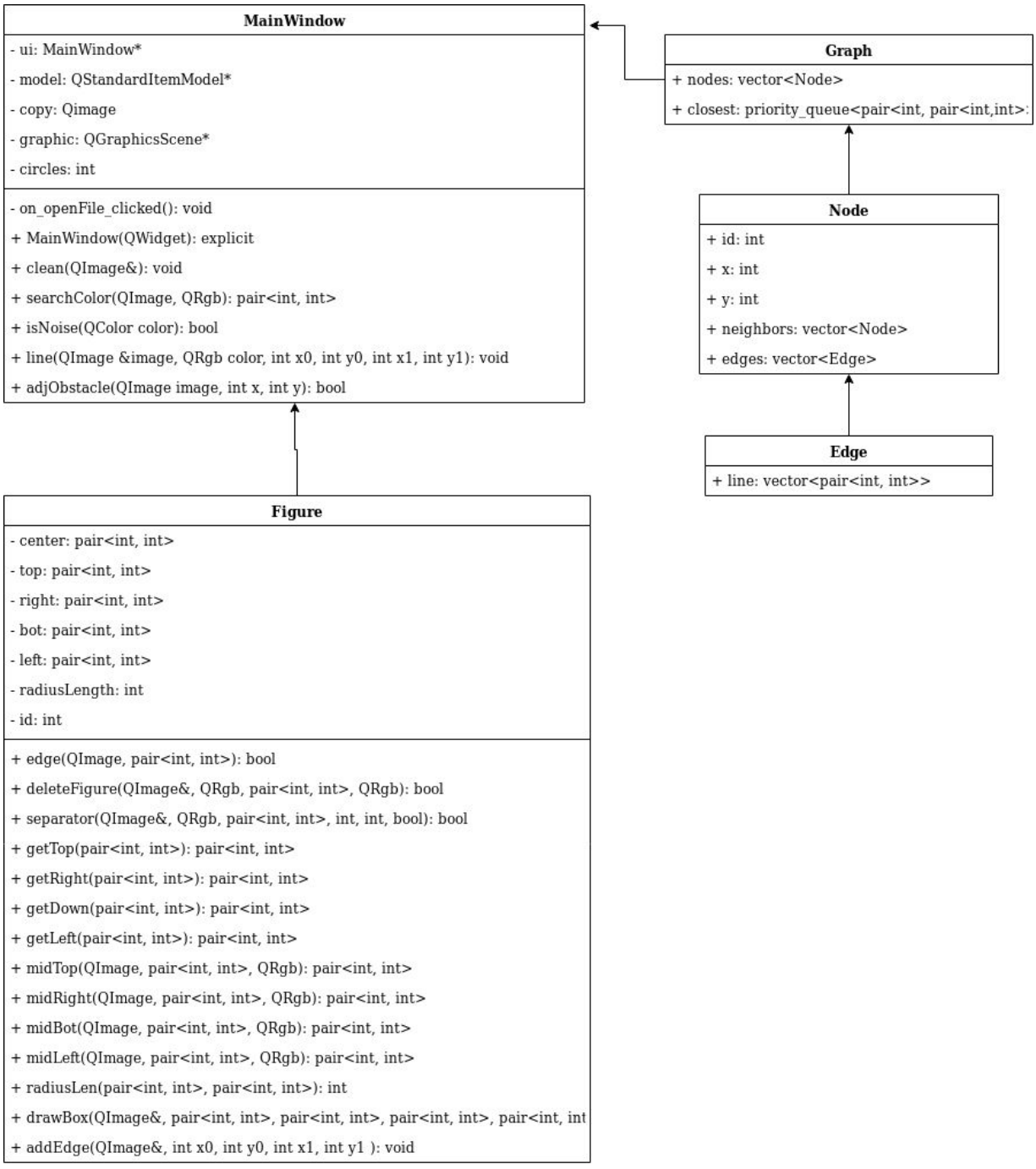
Fuerza bruta

Jesús Uriel Guzmán Mendoza

I. Diagrama de clases propuesto



II. Objetivo

El objetivo de esta actividad es es, ya con los los círculos que han sido detectados en la etapa de localización de círculo, crear conecciones entre cada uno de estos nodos, tal que no se encuentren figuras que obstruyan en la conexión de los círculos. La idea es crear un grafo en donde cada círculo sea un vértice y cada conexión sea una arista, cualquier figura que no sea una arista es un obstáculo, y por lo tanto, impide la conexión de un vértice a otro. Se encuentra el par de puntos más cercanos y se permite ordenar a los círculos de tal forma que estén descendientemente por criterio de los radios.

III. Marco teórico

Grafo

En matemáticas y ciencias de la computación un grafo es un conjunto de objetos llamados vértices o nodos unidos por enlaces llamados aristas o arcos, que permiten representar las relaciones binarias entre elementos de un conjunto.

Un grafo se representa como un conjunto de puntos unidos por líneas.

Los grafos permites representar las interrelaciones entre unidades que interactúan unas con otras. Prácticamente cualquier problema puede representarse mediante un grafo, y su estudio transcience a diversas áreas de las ciencias exactas y las ciencias sociales.

Distancia euclideana

La distancia euclideana es la distancia “ordinaria” entre dos puntos utilizando el teorema de pitágoras. En un espacio bidimensional, la distancia P_1 y P_2, de coordenadas cartesianas (x_1, y_1) y (x_2, y_2) respectivamente, es:

$$d_E(P_1, P_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Esta misma fórmula nos sirve para saber muchas cosas en esta actividad, la obvia: la distancia entre dos puntos (para saber el par de puntos más cercanos) y otra: si las coordenadas de un click de un cursor pertenecen a algún círculo. Esto es; si la distancia es menor o igual al radio de un círculo.

Fuerza bruta

Es un paradigma algorítmico general que consiste en sistemáticamente enumerar todos los posibles candidatos para la solución y revisar si cada candidato satisface el enunciado del problema. En esta actividad se utilizará este método de resolución dos veces: la primera es para saber si cada nodo satisface la condición de estar conectado con otro vértice si no hay obstáculos, y la otra es para encontrar el par de puntos más cercanos.

IV. Desarrollo

Desarrollando el grafo

Partiendo desde la localización de los círculos, el siguiente paso a desarrollar sería el de crear un grafo. Para esto, se ha creado un vector de nodos visitados, un índice, y un modelo de n x n para la matriz de adyacencia. Se itera desde 0 hasta el número de vértices que ya se saben que tendrá el grafo, y se le asigna un identificador a cada nodo con respecto a este ciclo. Los nodos reciben las posiciones de sus centros que ya han sido guardadas anteriormente, y se ejecuta un segundo ciclo de 0 hasta n con el objetivo de encontrar potenciales nodos vecinos.

De primero, se asume que todos son vecinos y se genera un algoritmo de fuerza bruta donde decimos que todos los nodos podrían conectados al menos que se pruebe lo contrario. Cuando se hace esta iteración de fuerza bruta, no nos interesa empatar un nodo que tenga el mismo identificador con otro ya que aquí no existen los lazos, también se revisa que sus identificadores no estén en la basura para la eliminación de vértices, de lo cual se hablará más adelante. Después de la primera iteración, mandamos el nodo i a el conjunto de nodos visitados ya que no nos interesa iterar sobre el mismo nodo dos veces (todos los caminos son bi-direccionales, por lo tanto si un nodo es vecino de otro, el otro también es vecino del nodo) y creamos un objeto Edge. Este objeto Edge contiene un atributo Line, el cual es un vector de coordenadas que nos indican el camino de un vértice a otro. Para rellenar este vector de coordenadas, se llama a una función addEdge, la cual realiza una búsqueda de obstáculos. Si no existe nada obstruyendo la conexión entre los vértices, entonces al nodo *i* se le asigna una arista y decimos que el nodo *j* es vecino del nodo *i,* y por lo tanto, comparten esta arista. A la misma vez, se va generando una matriz de adyacencia.

La complejidad algoritmica del construcción de este grafo es de O(n^2).

Descubriendo obstáculos

Para construir el vector de coordenadas que pertenecen a la arista de un nodo a otro, se llama a la función addEdge. Esta función lo que hace es que recibe con parámetro los centros de los dos nodos que se quiere buscar su conectabilidad, y se traza una línea recta utilizando el algoritmo de Bresenham (Apéndice A). Para saber si hay un obstáculo, se requiere saber cuántas veces se ha entrado y salido de diferentes figuras (FIG. 22). Decimos que si el número de veces que se ha salido de una figura es mayor o igual a 1, entonces hay un obstáculo y si es igual a 1, ha sido una conexión exitosa (ya que se requiere salir de una figura al menos una vez para entrar a otra). En el caso de haber encontrado un obstáculo, devolvemos el vector con valores de -1 para indicar que no se pudo conectar exitosamente. En el caso contrario, se regresan todas las coordenadas.

En el caso de que no se hayan encontrado obstáculos, se manda a llamar la función line(), que viene siendo básicamente lo mismo que addEdge(), pero en este ya se tiene la seguridad de que es una conexión y por lo tanto va a dibujar la arista de un color.

Par de nodos más cercanos

Para este algoritmo, se requiere saber x0, y0, x1, y1, donde {x0, y0} pertenecen al centro de algún, nodo, y {x1, y1} pertenecen al centro de algún otro nodo. Mediante otro algoritmo de fuerza bruta, a cada comparación de le saca la distancia utilizando la fórmula d = raíz((x1-x0)^2 - (y1-y0)^2) y se empuja el par ordenado a un priority_queue perteneciente a la clase Graph. También se guarda que el par ya fue visitado en un set para no tener el mismo resultado dos veces. El priority_queue es muy útil ya que ordena automáticamente los pares con respecto al mapeado de sus distancias, haciendo que para saber cuál es el par ordenado más cercano, insertamos los números en negativo (para ordenar descendentemente negativamente) y luego multiplicar cada valor por -1, lo cual crea un ordenamiento de forma ascendente; a este le sacamos el método de top() para saber el par ordenado con la distancia más pequeña. Después, como se conoce el centro de estos nodos ya que fueron guardados en la práctica anterior, reutilizamos la funcón de pintar círculos para pintarlos de un color que represente su cercanía.

Su complejidad algoritmica es de O(n^2).

Eliminación de vértices

(FIG. 19, FIG. 20 y FIG. 21) Para esto, se utilizó la función de Qt *mousepressevent,* la cual nos dice las coordenadas relativas con respecto a la imagen utilizando la clase *QPoint*. En el mainwindow se tiene un registro de cual coordenada fue marcada para la eliminación, y en una función, determinamos si la coordenada pertenece o no pertenece a un nodo. Si no pertenece, no se hace nada, si pertenece a un nodo activo, entonces se manda su id a la basura y este no puede ser recuperado hasta que se vuelva a cargar otra vez la imagen. El atributo de basura marca a los nodos como “ignorados” y se vuelve a correr el algoritmo sin tomar en cuenta a estos nodos en la basura.

Para saber si una coordenada pertenece o no pertenece a un nodo, averiguamos {xp, yp} como la coordenada regresada del cursor, {xc, yc} son las coordenadas de algún nodo, r es el radio de el *iésimo* nodo y d la distancia. Decimos que si la distancia d = raíz((xp-xc)^2 + (yp-yc)^2) es menor al radio del iésimo nodo, entonces la coordenada pertenece a ese vértice y por lo tanto será mandado a la basura.

La eliminación de los vérticos se ve instantáneamente en el display de la tabal de información y la matriz de adyacencia (FIG. 18).

Si existía un vértice que estaba obstruyendo la conexión a otro vértice, la eliminación de este vértice ocasiona que los nodos que no se podían conectar por culpa de este vértice se vuelvan a conectar. Esto también se ve reflejado inmediatamente en las tablas.

Ordenamiento de radios

(FIG. 16 y FIG. 17) El ordenamiento de los radios se lleva al cabo creando una estructura *compare* y creando una función booleana aprovechando el operador *()* que nos brinda std::sort, en la función se reciben dos nodos y el output de la función es verdadero si el radio del primero nodo es mayor al radio del segundo nodo y falso si son iguales o el segundo radio es mayor al primer radio. Se crea una copia del grafo para no modificar el original y se vuelve a llamar la función que genera la tabla. El botón para ordenar es un *toggle*, por lo tanto existe un estado de ordenado y un estado de no-ordenado, como un prendido y apagado.

Su complejidad algorítmica es de O(n^2).

V. Pruebas y resultados

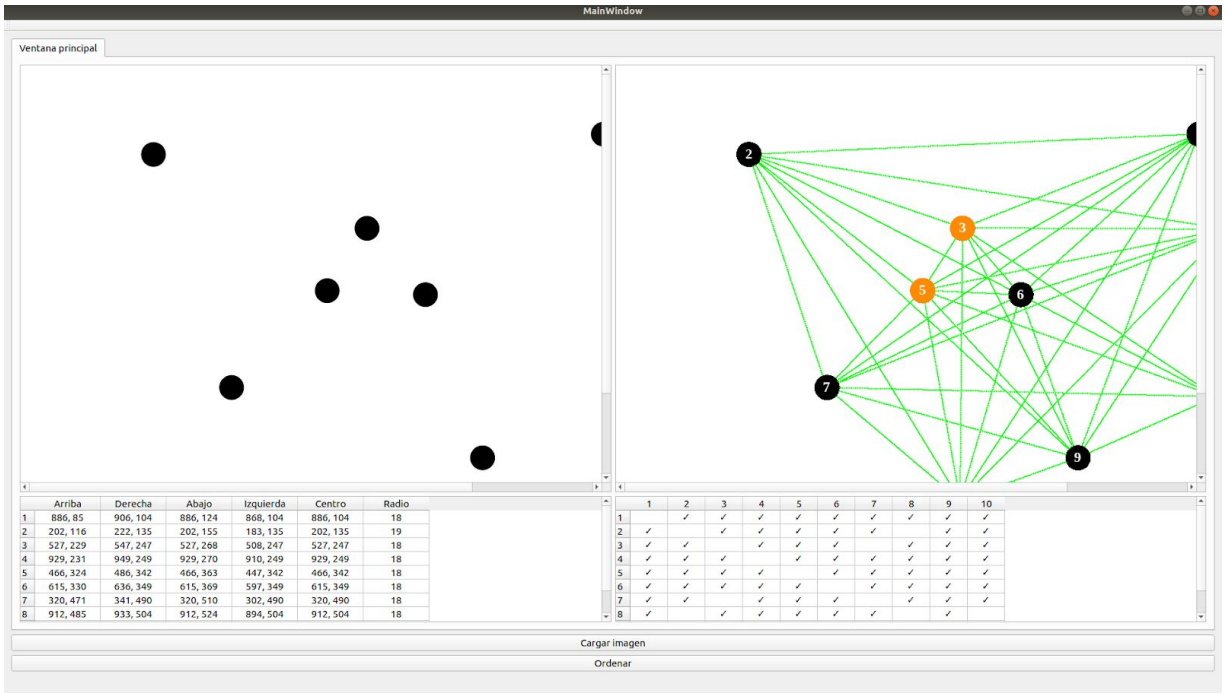
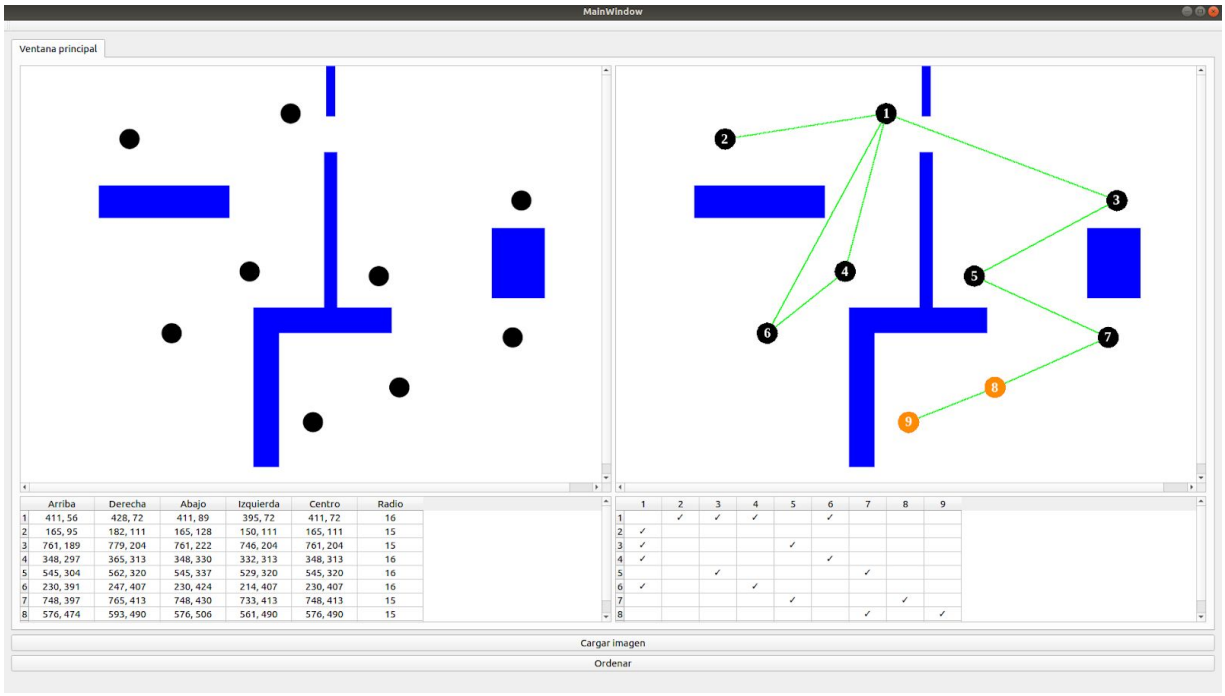


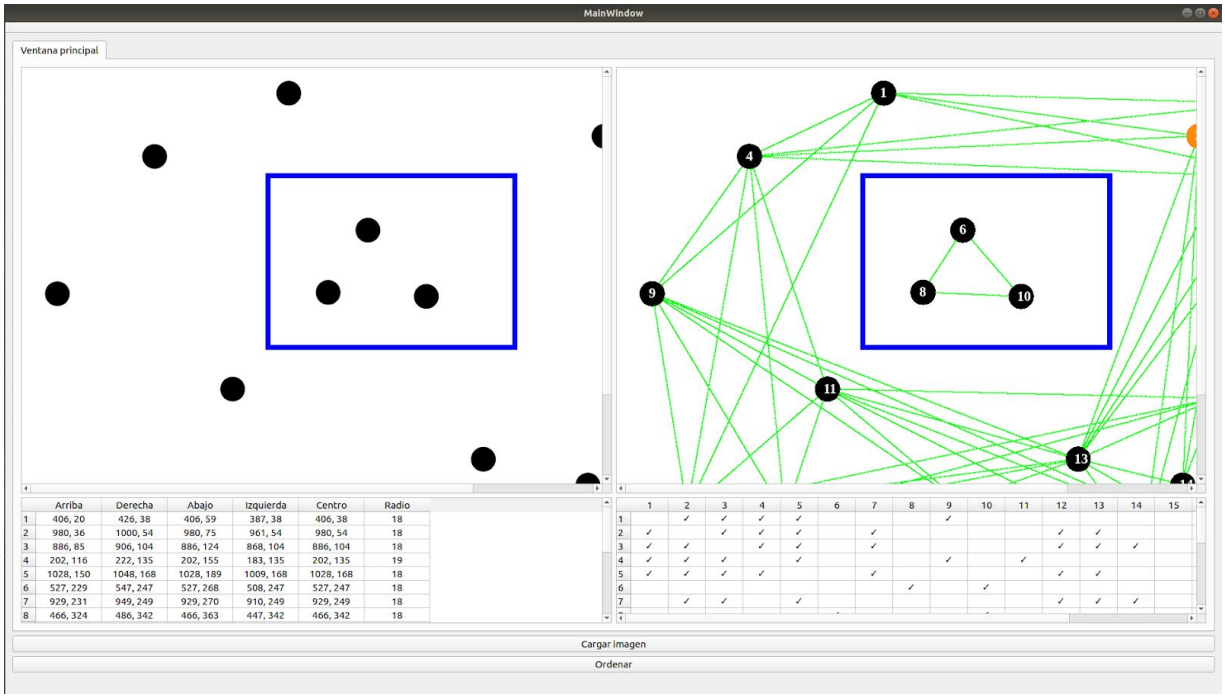
FIG. 1



	Arriba	Derecha	Abajo	Izquierda	Centro	Radio
1	411, 56	428, 72	411, 89	395, 72	411, 72	16
2	165, 95	182, 111	165, 128	150, 111	165, 111	15
3	761, 189	779, 204	761, 222	746, 204	761, 204	15
4	348, 297	365, 313	348, 330	332, 313	348, 313	16
5	545, 304	562, 320	545, 337	529, 320	545, 320	16
6	230, 391	247, 407	230, 424	214, 407	230, 407	16
7	748, 397	765, 413	748, 430	733, 413	748, 413	15
8	576, 474	593, 490	576, 506	561, 490	576, 490	15

	1	2	3	4	5	6	7	8	9
1		✓	✓	✓		✓			
2	✓								
3	✓				✓				
4	✓					✓			
5			✓				✓		
6	✓			✓					
7					✓			✓	
8							✓		✓

FIG. 2



	Arriba	Derecha	Abajo	Izquierda	Centro	Radio
1	406, 20	426, 38	406, 59	387, 38	406, 38	18
2	980, 36	1000, 54	980, 75	961, 54	980, 54	18
3	886, 85	906, 104	886, 124	868, 104	886, 104	18
4	202, 116	222, 135	202, 155	183, 135	202, 135	19
5	1028, 150	1048, 168	1028, 189	1009, 168	1028, 168	18
6	527, 229	547, 247	527, 268	508, 247	527, 247	18
7	929, 231	949, 249	929, 270	910, 249	929, 249	18
8	466, 324	486, 342	466, 363	447, 342	466, 342	18

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1		✓	✓	✓	✓				✓						
2	✓		✓	✓	✓		✓					✓	✓		
3	✓	✓		✓	✓		✓					✓	✓	✓	
4	✓	✓	✓		✓				✓		✓				
5	✓	✓	✓	✓			✓					✓	✓		
6								✓		✓					
7		✓	✓		✓							✓	✓	✓	

FIG. 3

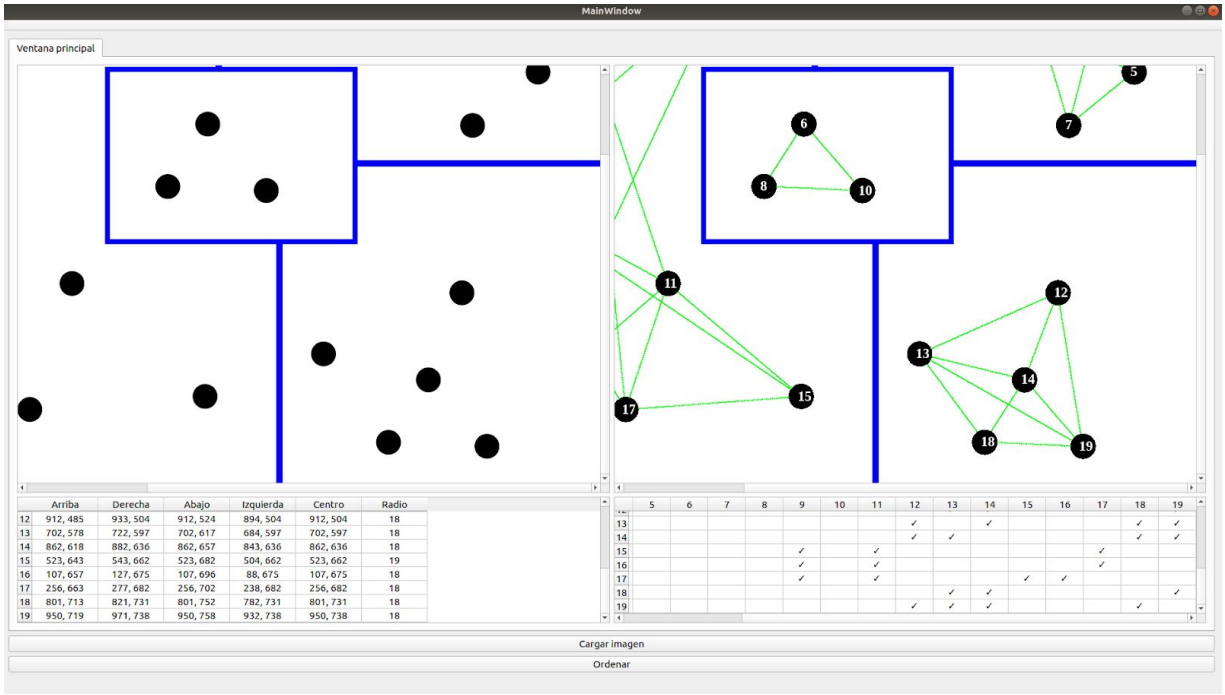

MainWindow
Ventana principal
Arriba Derecha Abajo Izquierda Centro Radio
12 912, 485 933, 504 912, 524 894, 504 912, 504 18
13 702, 578 722, 597 702, 617 684, 597 702, 597 18
14 862, 618 882, 636 862, 657 843, 636 862, 636 18
15 523, 643 543, 662 523, 682 504, 662 523, 662 19
16 107, 657 127, 675 107, 696 88, 675 107, 675 18
17 256, 663 277, 682 256, 702 238, 682 256, 682 18
18 801, 713 821, 731 801, 752 782, 731 801, 731 18
19 950, 719 971, 738 950, 758 932, 738 950, 738 18
5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
Cargar imagen
Ordenar

FIG. 4

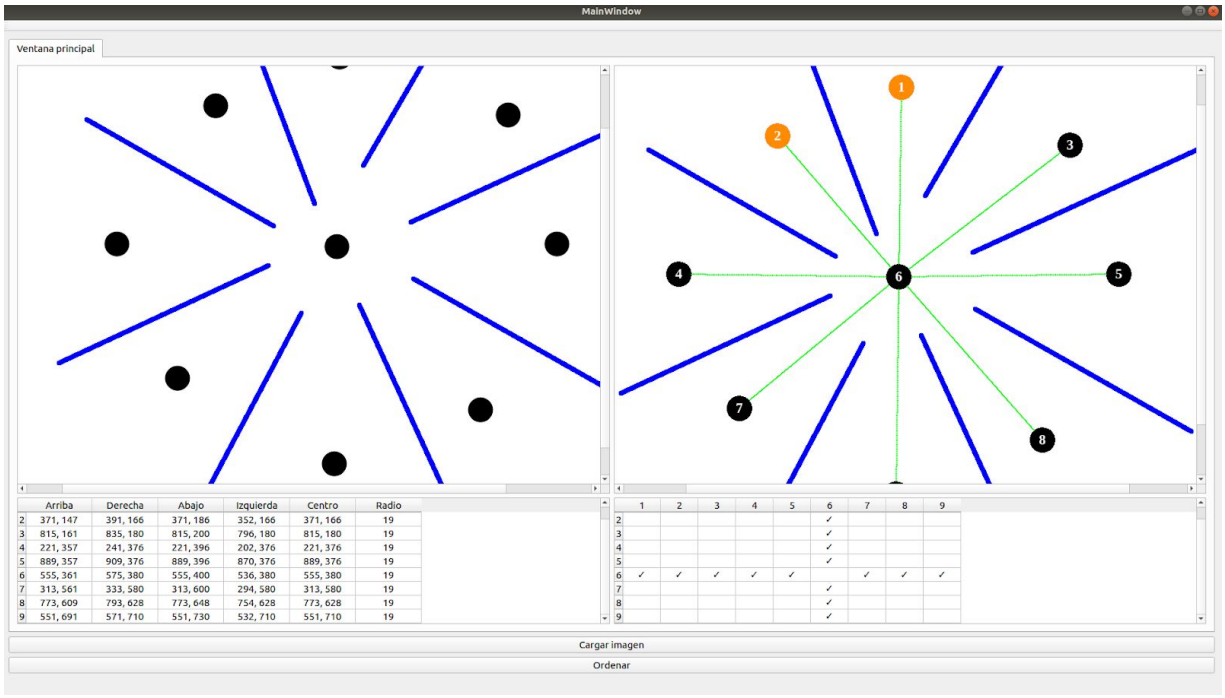

MainWindow
Ventana principal
Arriba Derecha Abajo Izquierda Centro Radio
2 371, 147 391, 166 371, 186 352, 166 371, 166 19
3 815, 161 835, 180 815, 200 796, 180 815, 180 19
4 221, 357 241, 376 221, 396 202, 376 221, 376 19
5 889, 357 909, 376 889, 396 870, 376 889, 376 19
6 555, 361 575, 380 555, 400 536, 380 555, 380 19
7 313, 561 333, 580 313, 600 294, 580 313, 580 19
8 773, 609 793, 628 773, 648 754, 628 773, 628 19
9 551, 691 571, 710 551, 730 532, 710 551, 710 19
1 2 3 4 5 6 7 8 9
Cargar imagen
Ordenar

FIG. 5

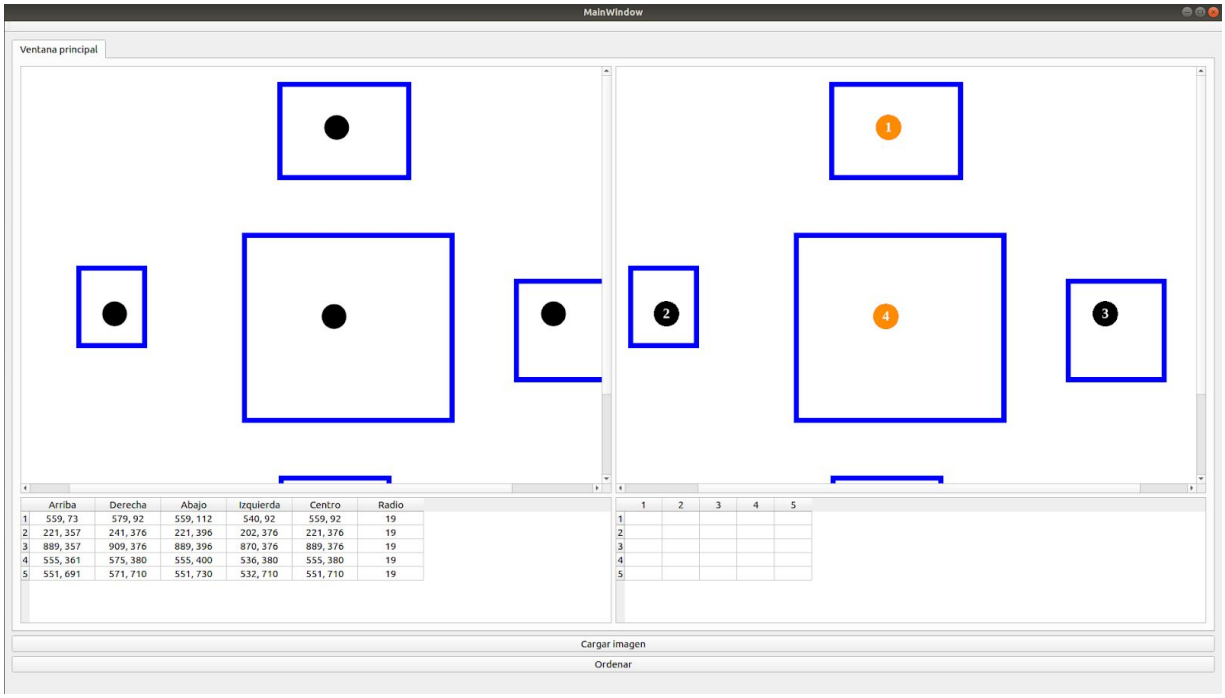

MainWindow
Ventana principal
Arriba Derecha Abajo Izquierda Centro Radio
1 559, 73 579, 92 559, 112 540, 92 559, 92 19
2 221, 357 241, 376 221, 396 202, 376 221, 376 19
3 889, 357 909, 376 889, 396 870, 376 889, 376 19
4 555, 361 575, 380 555, 400 536, 380 555, 380 19
5 551, 691 571, 710 551, 730 532, 710 551, 710 19
1 2 3 4 5
Cargar imagen
Ordenar

FIG. 6

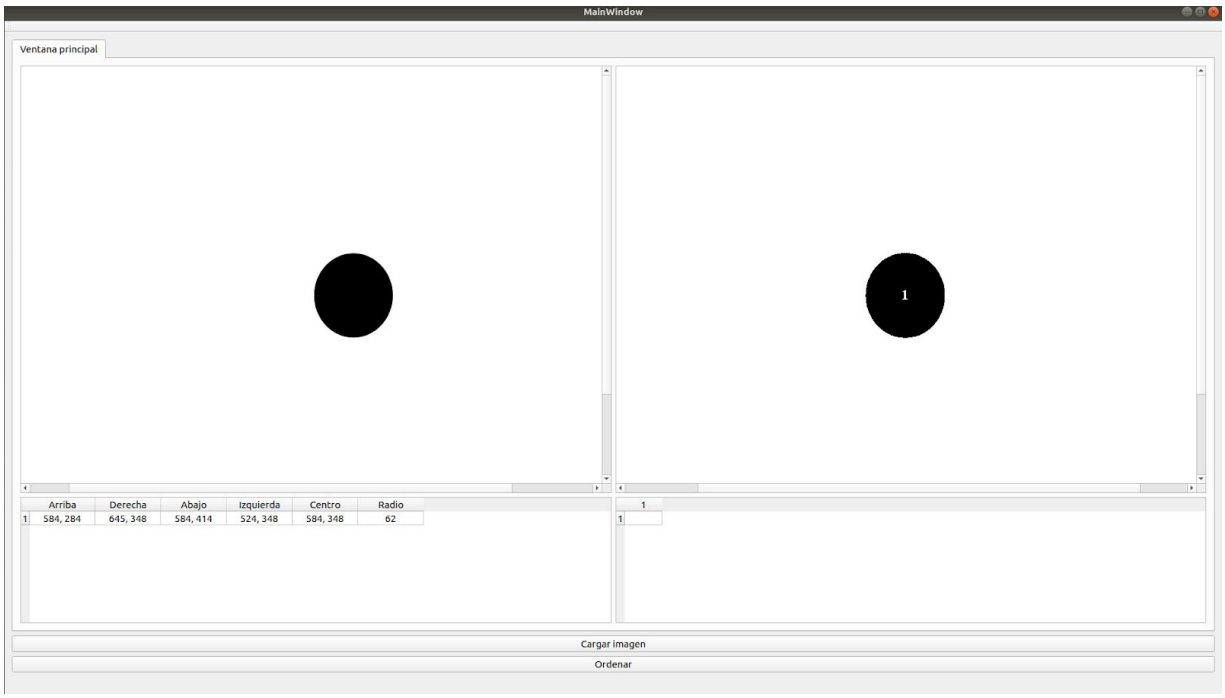

MainWindow
Ventana principal
1
Arriba
Derecha
Abajo
Izquierda
Centro
Radio
1
584, 284
645, 348
584, 414
524, 348
584, 348
62
1
Cargar imagen
Ordenar

FIG. 7

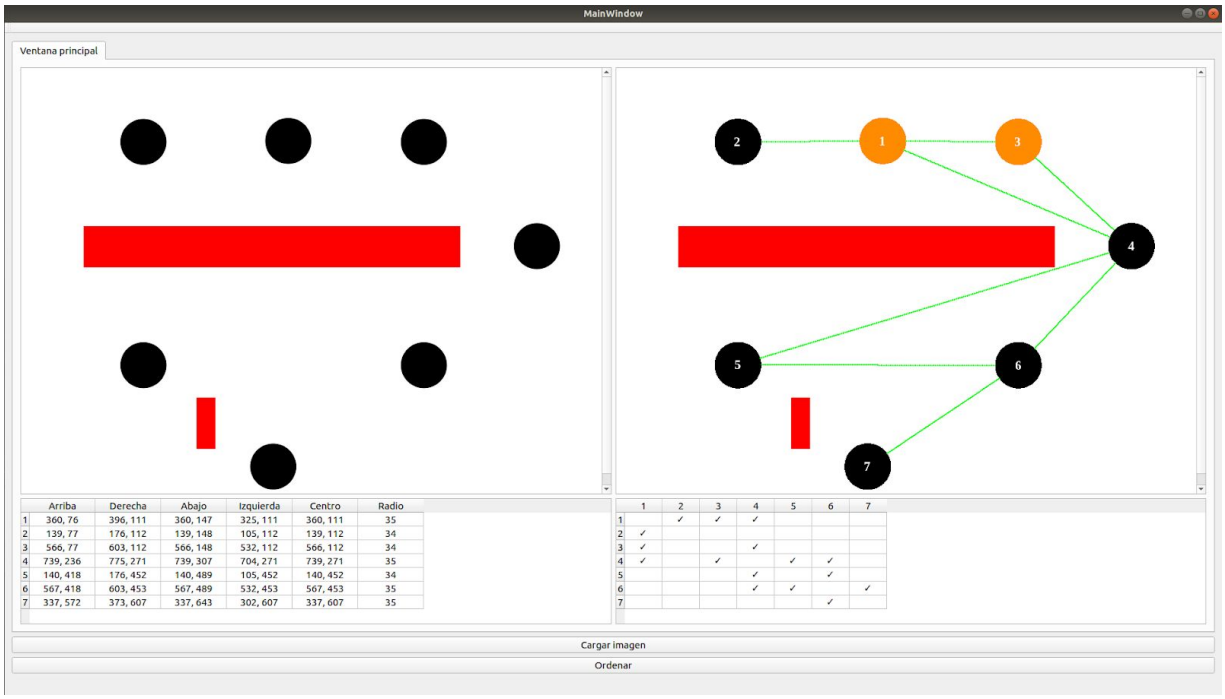

MainWindow
Ventana principal
2
1
3
4
5
6
7
Arriba Derecha Abajo Izquierda Centro Radio
1 360, 76 396, 111 360, 147 325, 111 360, 111 35
2 139, 77 176, 112 139, 148 105, 112 139, 112 34
3 566, 77 603, 112 566, 148 532, 112 566, 112 34
4 739, 236 775, 271 739, 307 704, 271 739, 271 35
5 140, 418 176, 452 140, 489 105, 452 140, 452 34
6 567, 418 603, 453 567, 489 532, 453 567, 453 35
7 337, 572 373, 607 337, 643 302, 607 337, 607 35
1 2 3 4 5 6 7
Cargar imagen
Ordenar

FIG. 8

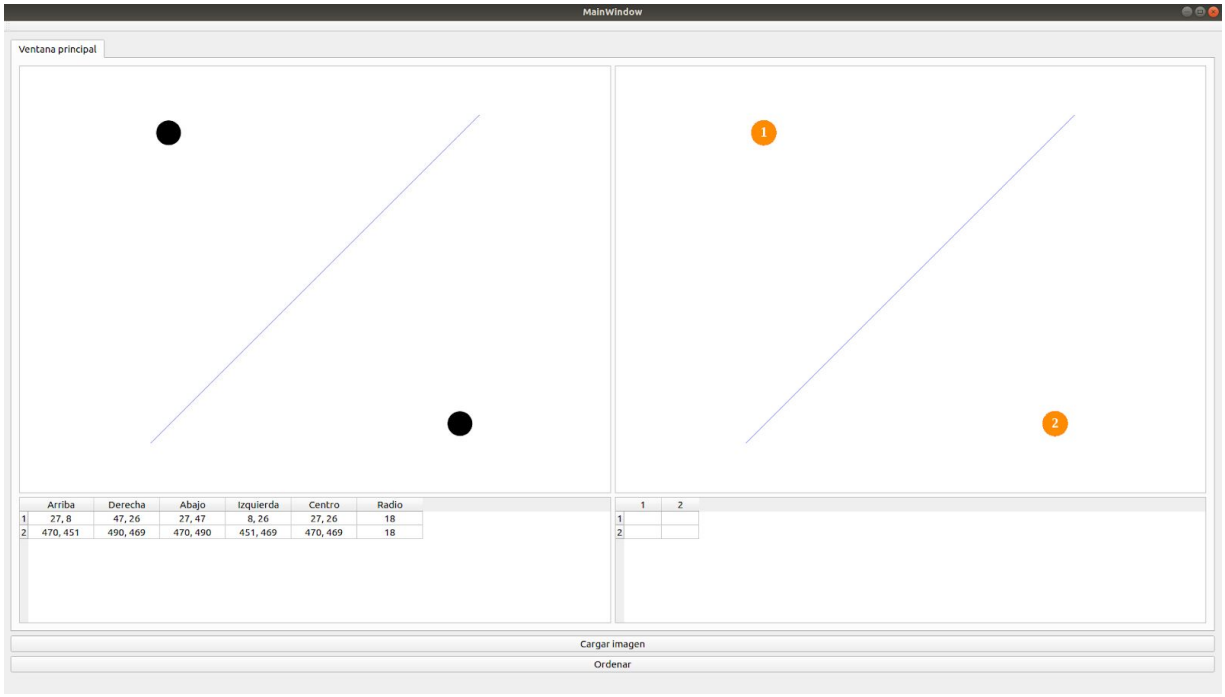

MainWindow
Ventana principal
1
2
Arriba Derecha Abajo Izquierda Centro Radio
1 27, 8 47, 26 27, 47 8, 26 27, 26 18
2 470, 451 490, 469 470, 490 451, 469 470, 469 18
1 2
Cargar imagen
Ordenar

FIG. 9

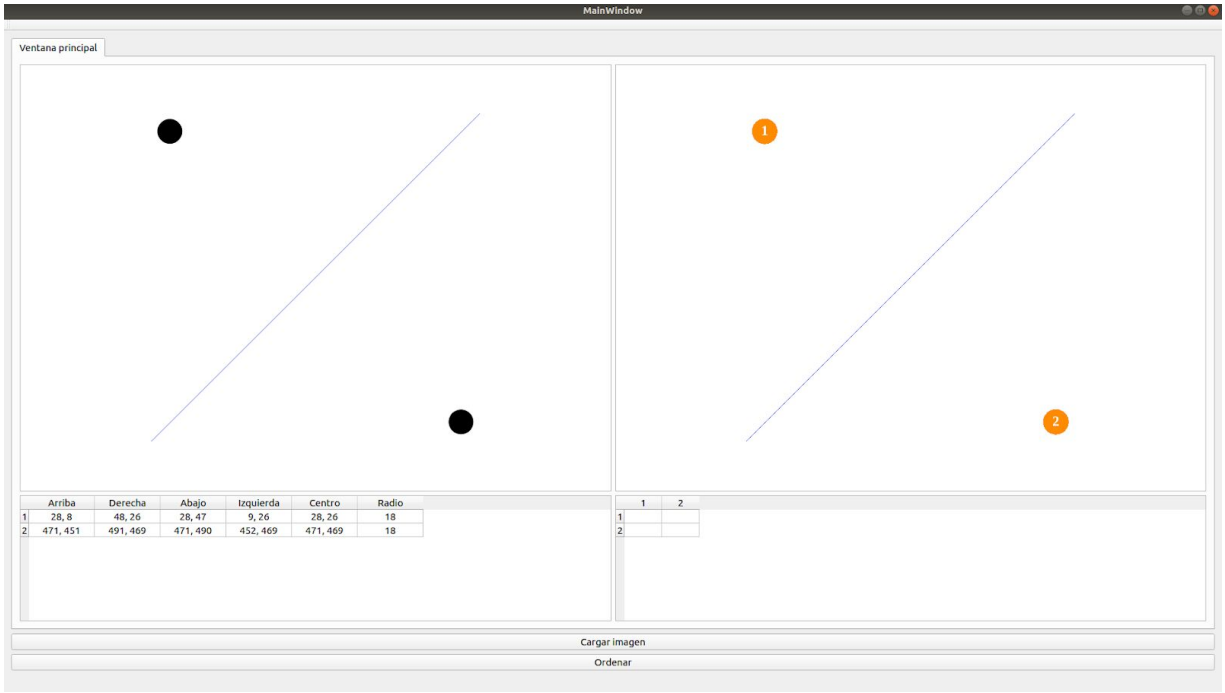

MainWindow
Ventana principal
1
2
Arriba
Derecha
Abajo
Izquierda
Centro
Radio
1
28, 8
48, 26
28, 47
9, 26
28, 26
18
2
471, 451
491, 469
471, 490
452, 469
471, 469
18
1
2
1
2
Cargar imagen
Ordenar

FIG. 10

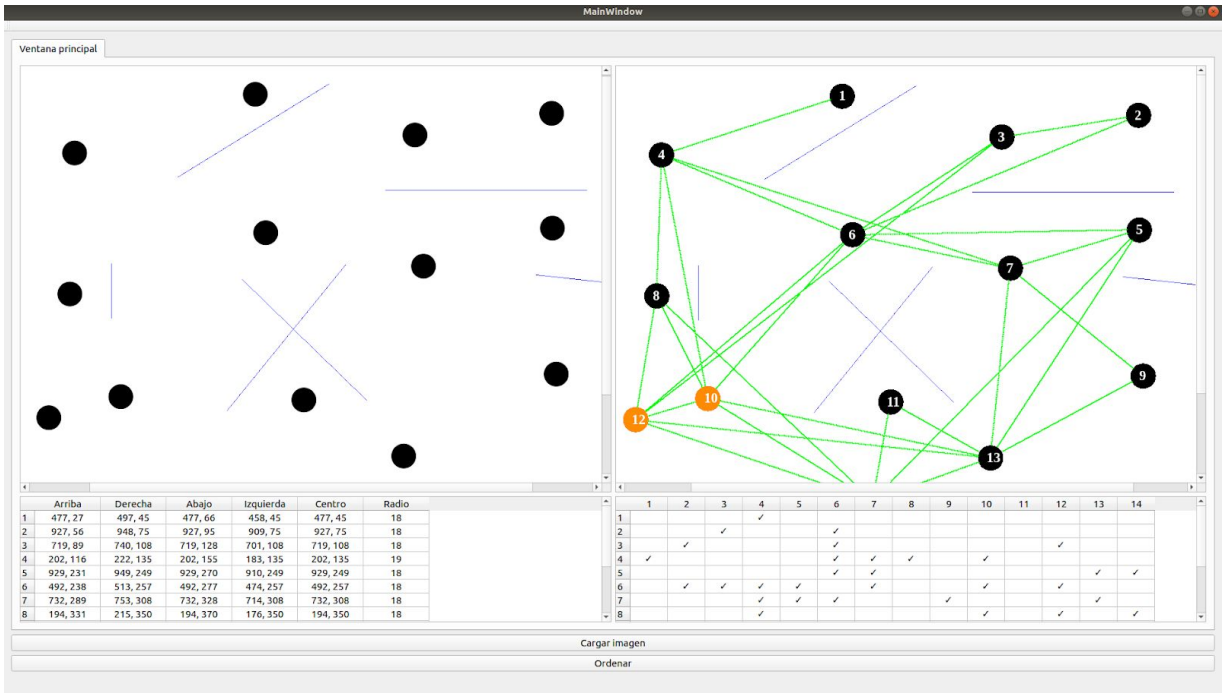

MainWindow
Ventana principal
1
2
3
4
5
6
7
8
9
10
11
12
13
Arriba Derecha Abajo Izquierda Centro Radio
1 477, 27 497, 45 477, 66 458, 45 477, 45 18
2 927, 56 948, 75 927, 95 909, 75 927, 75 18
3 719, 89 740, 108 719, 128 701, 108 719, 108 18
4 202, 116 222, 135 202, 155 183, 135 202, 135 19
5 929, 231 949, 249 929, 270 910, 249 929, 249 18
6 492, 238 513, 257 492, 277 474, 257 492, 257 18
7 732, 289 753, 308 732, 328 714, 308 732, 308 18
8 194, 331 215, 350 194, 370 176, 350 194, 350 18
1 2 3 4 5 6 7 8 9 10 11 12 13 14
Cargar imagen
Ordenar

FIG. 11

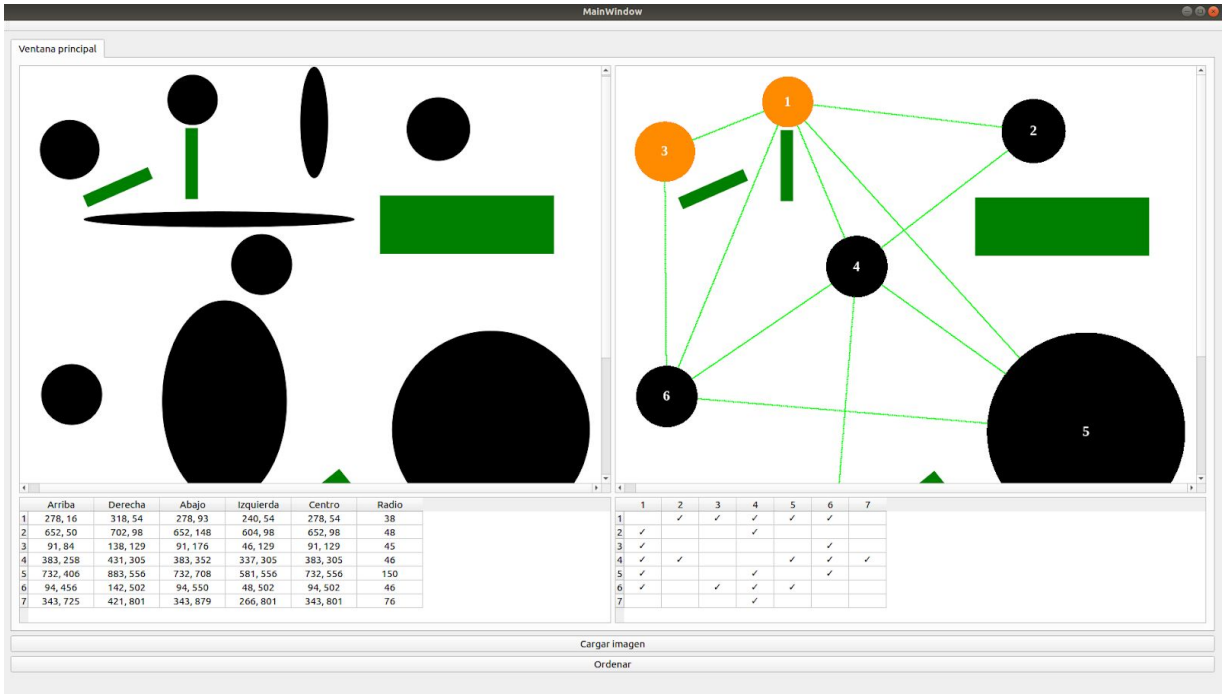

MainWindow
Ventana principal
1
2
3
4
5
6
Arriba Derecha Abajo Izquierda Centro Radio
1 278, 16 318, 54 278, 93 240, 54 278, 54 38
2 652, 50 702, 98 652, 148 604, 98 652, 98 48
3 91, 84 138, 129 91, 176 46, 129 91, 129 45
4 383, 258 431, 305 383, 352 337, 305 383, 305 46
5 732, 406 883, 556 732, 708 581, 556 732, 556 150
6 94, 456 142, 502 94, 550 48, 502 94, 502 46
7 343, 725 421, 801 343, 879 266, 801 343, 801 76
1 2 3 4 5 6 7
Cargar imagen
Ordenar

FIG. 12

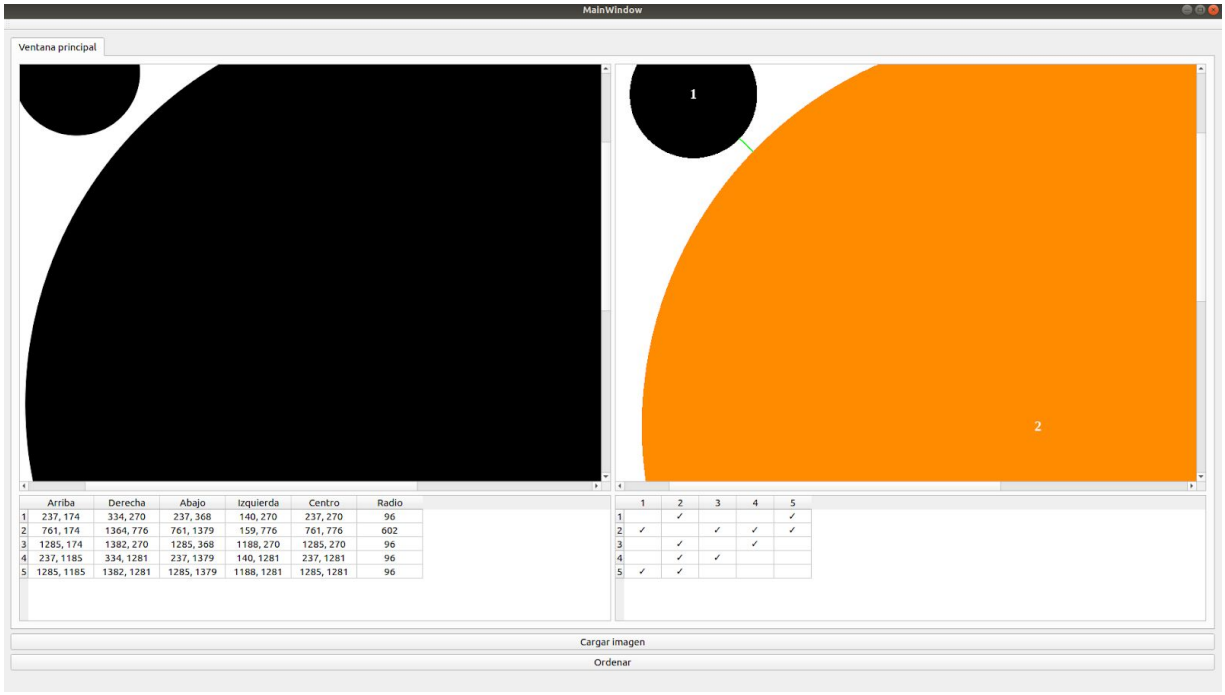

MainWindow
Ventana principal
1
2
Arriba Derecha Abajo Izquierda Centro Radio
1 237, 174 334, 270 237, 368 140, 270 237, 270 96
2 761, 174 1364, 776 761, 1379 159, 776 761, 776 602
3 1285, 174 1382, 270 1285, 368 1188, 270 1285, 270 96
4 237, 1185 334, 1281 237, 1379 140, 1281 237, 1281 96
5 1285, 1185 1382, 1281 1285, 1379 1188, 1281 1285, 1281 96
1 2 3 4 5
1 ✓ ✓
2 ✓ ✓ ✓ ✓
3 ✓ ✓
4 ✓ ✓
5 ✓ ✓
Cargar imagen
Ordenar

FIG. 13

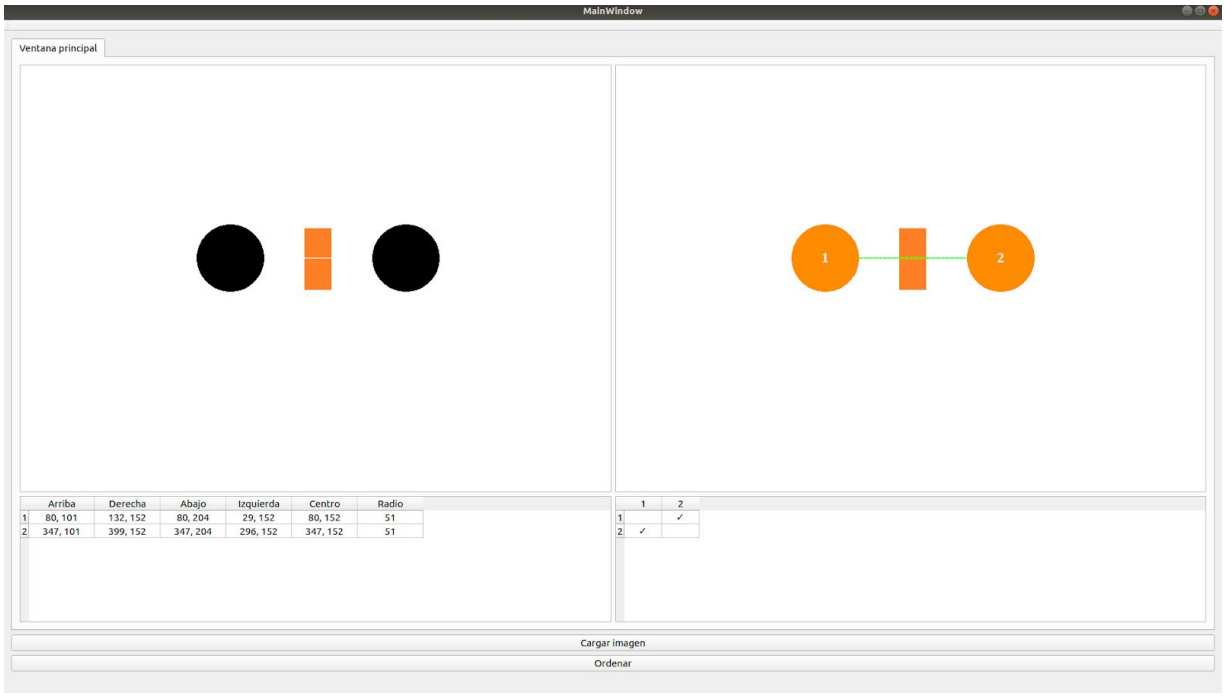

MainWindow
Ventana principal
1
2
Arriba Derecha Abajo Izquierda Centro Radio
1 80, 101 132, 152 80, 204 29, 152 80, 152 51
2 347, 101 399, 152 347, 204 296, 152 347, 152 51
1 2
1 ✓
2 ✓
Cargar imagen
Ordenar

FIG. 14


MainWindow
Ventana principal
1
2
Arriba Derecha Abajo Izquierda Centro Radio
1 80, 101 132, 152 80, 204 29, 152 80, 152 51
2 347, 101 399, 152 347, 204 296, 152 347, 152 51
1 2
1 ✓
2 ✓
Cargar imagen
Ordenar

FIG. 15

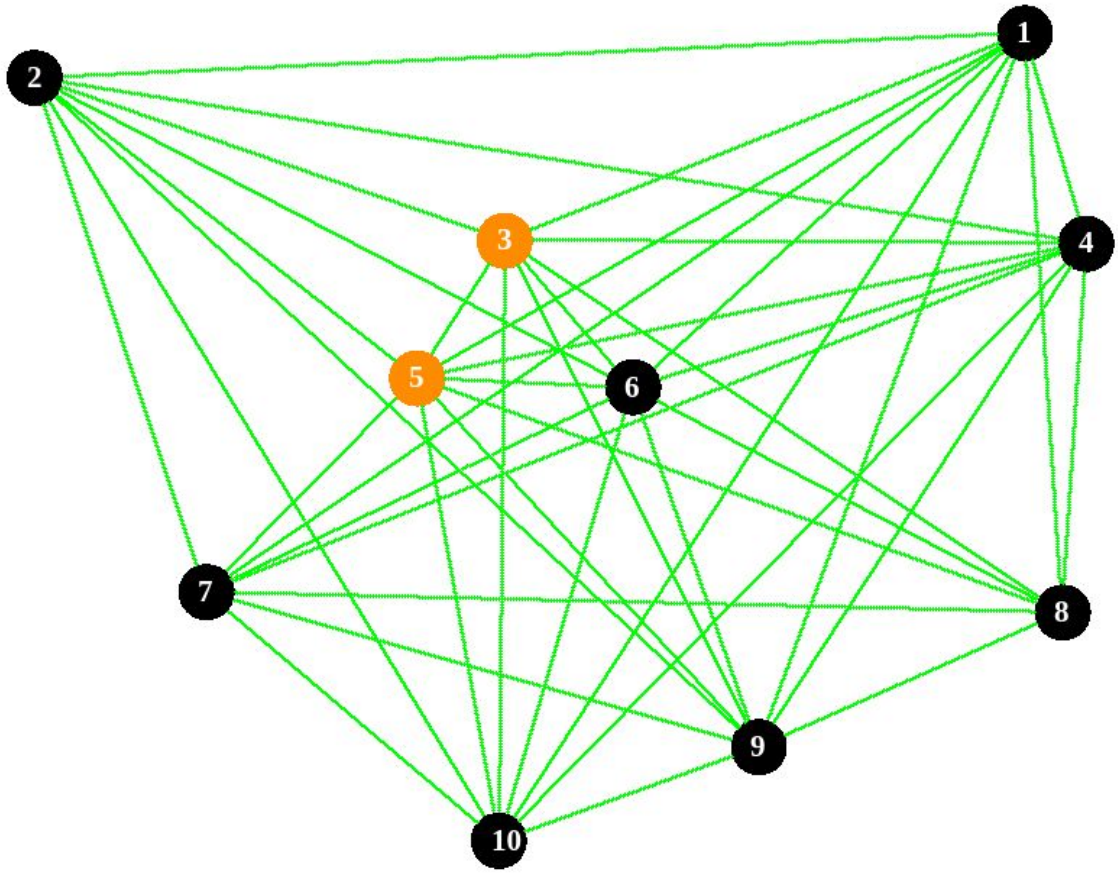
	Arriba	Derecha	Abajo	Izquierda	Centro	Radio
1	237, 174	334, 270	237, 368	140, 270	237, 270	96
2	761, 174	1364, 776	761, 1379	159, 776	761, 776	602
3	1285, 174	1382, 270	1285, 368	1188, 270	1285, 270	96
4	237, 1185	334, 1281	237, 1379	140, 1281	237, 1281	96
5	1285, 1185	1382, 1281	1285, 1379	1188, 1281	1285, 1281	96

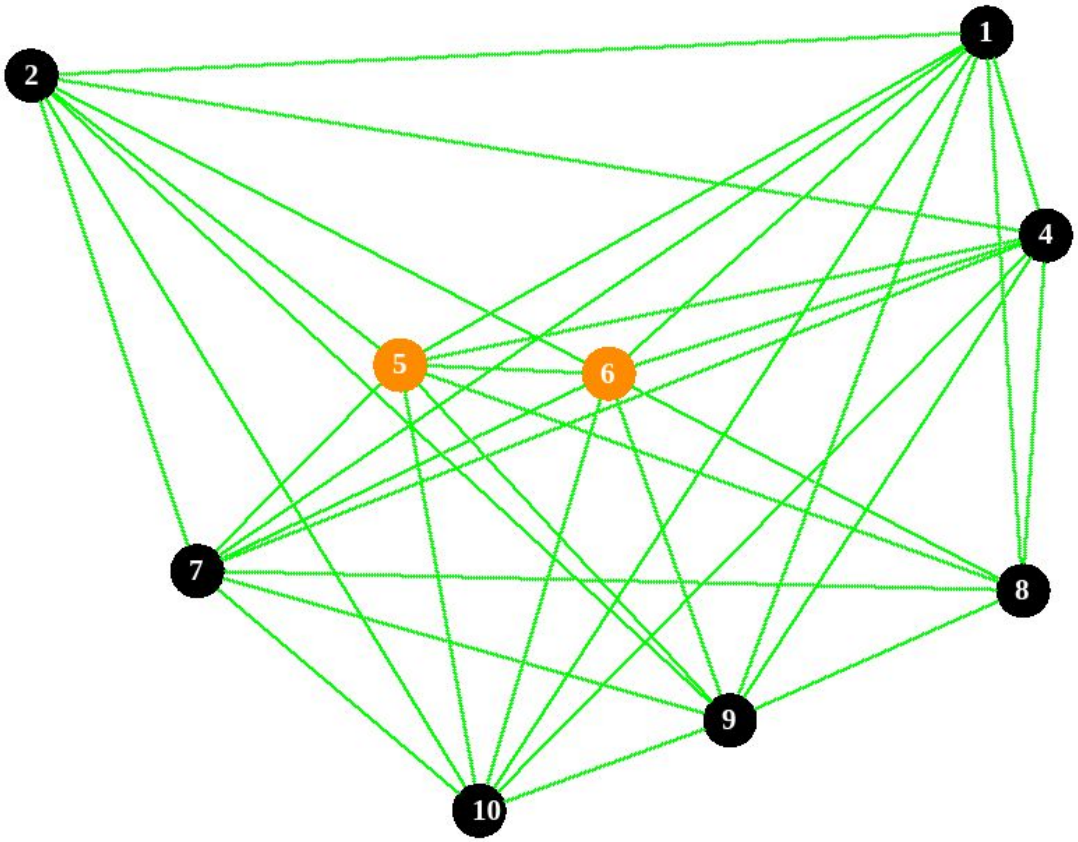
	Arriba	Derecha	Abajo	Izquierda	Centro	Radio
2	761, 174	1364, 776	761, 1379	159, 776	761, 776	602
1	237, 174	334, 270	237, 368	140, 270	237, 270	96
3	1285, 174	1382, 270	1285, 368	1188, 270	1285, 270	96
4	237, 1185	334, 1281	237, 1379	140, 1281	237, 1281	96
5	1285, 1185	1382, 1281	1285, 1379	1188, 1281	1285, 1281	96

FIG. 16 y FIG. 17

	1	2	3	4	5
1		✓			✓
2	✓		✓	✓	✓
3		✓		✓	
4		✓	✓		
5	✓	✓			

FIG. 18


1
2
3
4
5
6
7
8
9
10


1
2
4
5
6
7
8
9
10

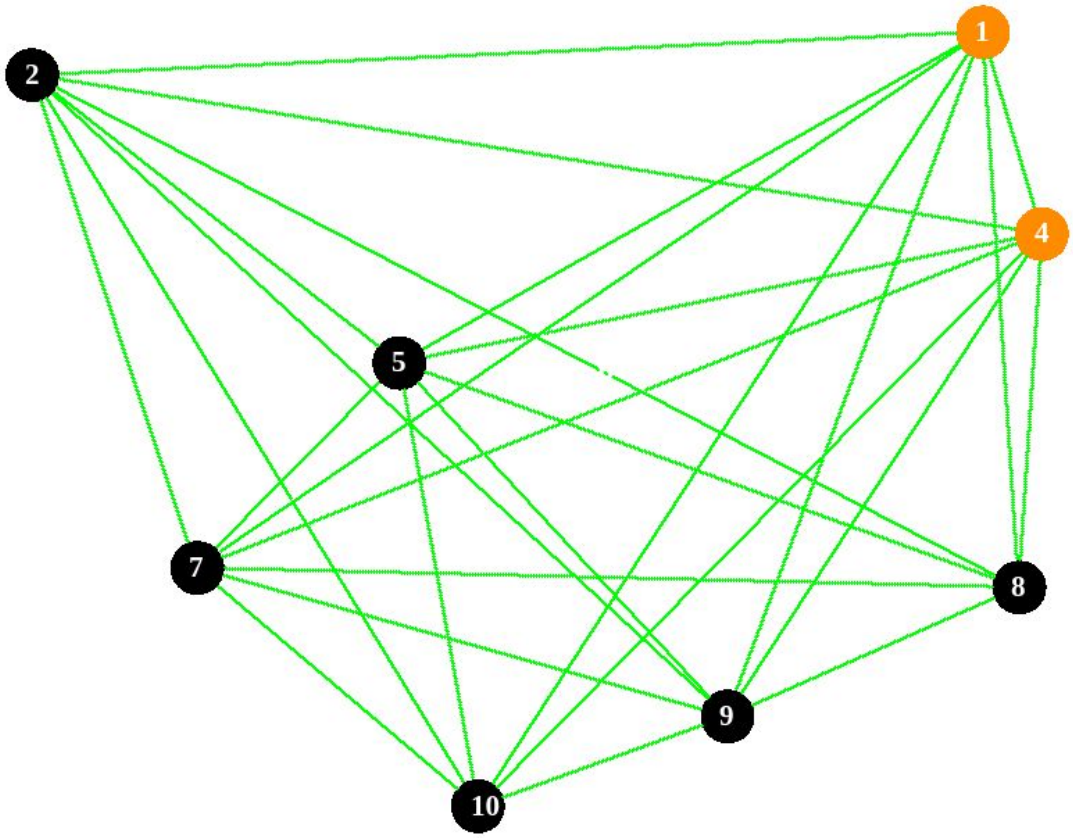


FIG. 19, FIG. 20, y FIG. 21

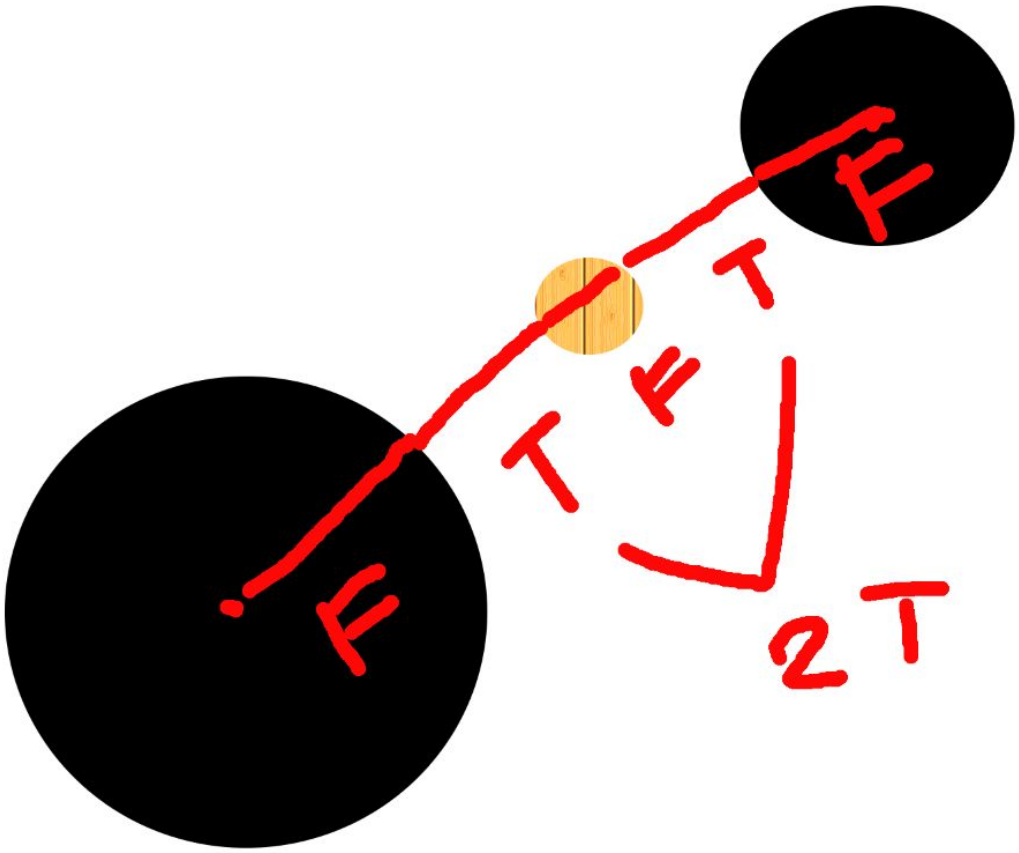


FIG. 22

VI. Conclusiones

Aunque un algoritmo de fuerza bruta es fácil de implementar, y siempre encontrará la solución si es que existe, su costo es proporcional al número de candidatos para una solución. Lo cual en varios problemas puede crecer muy, muy rápidamente mientras el tamaño de el problema aumenta. Por lo tanto, solamente veo razonable utilizar algoritmos de fuerza bruta cuando el tamaño del problema es reducido (como en el caso de esta actividad, a lo mucho, exisisten no más de 20 vértices, por lo cual son bastante rápidos los algoritmos).

También puedo ver que este paradigma de la programación puede resulta a ser útil cuando se está más preocupado de la simplicidad de encontrar una solución sobre la eficiencia del algoritmo.

VII. Apéndice A

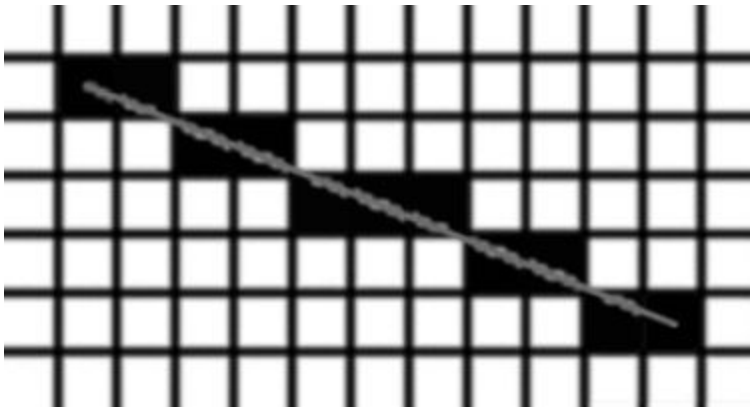


FIG. 23

El Algoritmo de Bresenham es un método rápido para el trazado de líneas en dispositivos gráficos, cuya cualidad más apreciada es que solo realiza cálculos con enteros.

Se puede adaptar para rasterizar también circunferencias y curvas. Los ejes verticales muestran las posiciones de rastreo y los ejes horizontales identifican columnas de pixel.

Actualmente se usa el nombre de algoritmos de Bresenham a toda una familia de algoritmos basado en modificaciones o ampliaciones del original aquí descrito.