UNIVERSIDAD DE GUADALAJARA

CENTRO UNIVERSITARIO DE CIENCIAS EXACTAS E INGENIERIAS

SEMINARIO DE SOLUCIÓN A PROBLEMAS DE ALGORITMIA

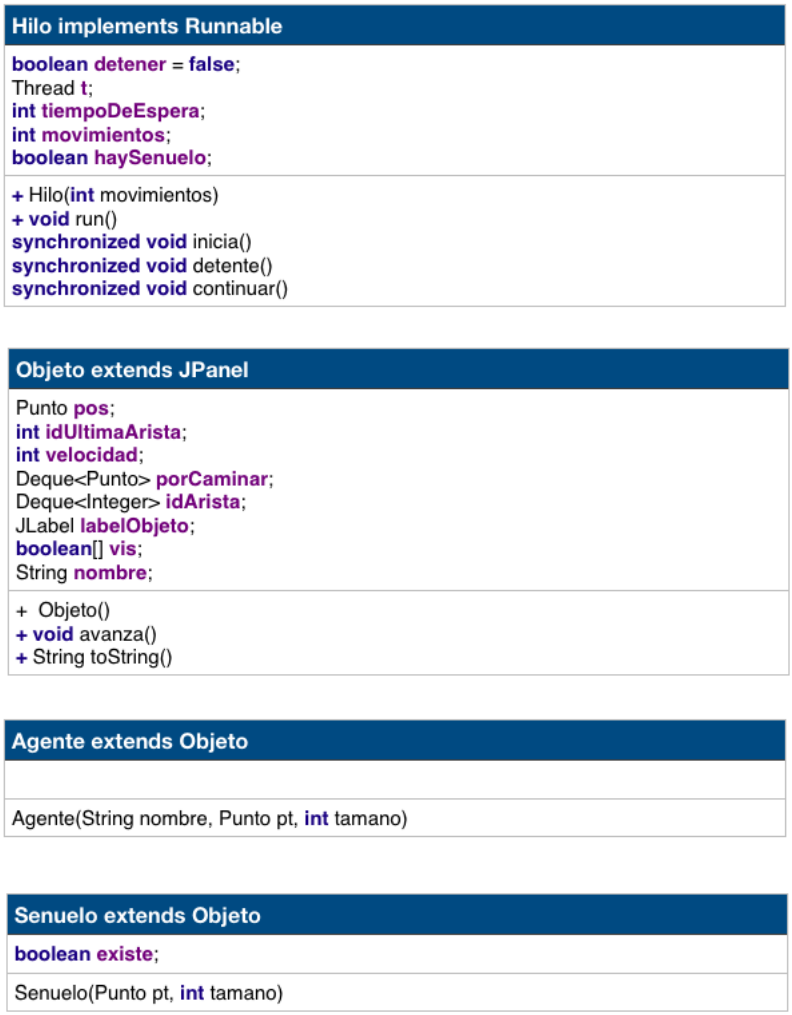
Actividad 3: Agentes en un grafo

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



2 Objetivo

Diseñar un sistema en base el sistema anterior capaz de crear un grafo, agregar agentes que se moverán a traves del grafo siempre siendo atraidos por el senuelo que se coloca (exactamente un senuelo), además se debe de poder ver el movimiento progresivo de los agentes mientras hacen el mejor recorrido por el grafo.

3 Marco teórico

Algoritmo A*:

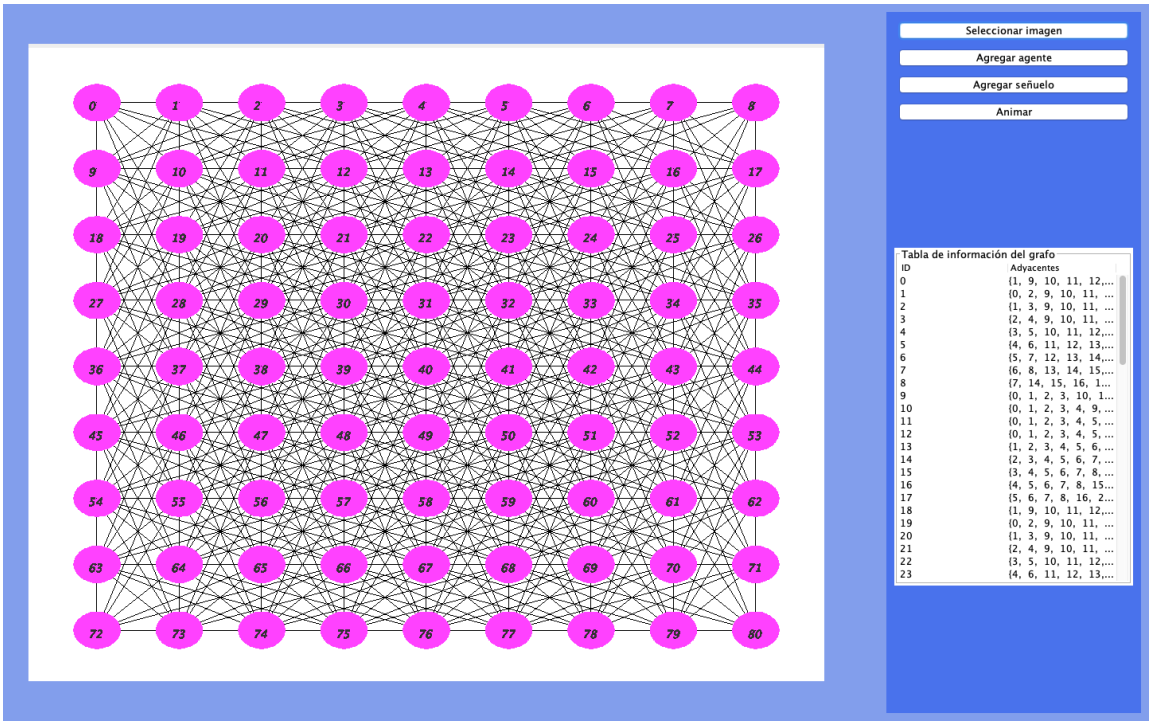
Es un algoritmo de búsqueda en grafo de tipo heurístico, este decide el mejor camino siempre y cuando se cumpla que la función f(x) = h(x) + g(x) es la menor de todas las posibles lo que le indica que es el mejor movimiento.
La función h(x) está denotada como la función heurística, en mi caso determinaba la distancia que separaba a los vértices si estos se movieran en línea rectas.
La función g(x) está denotada como la cantidad de puntos que se debe de desplazar para llegar de un (x0, y0) a otro (x2, y2) pasando por otro (x1, y1), está en una función que denota el coste real del desplazamiento.

4 Desarrollo

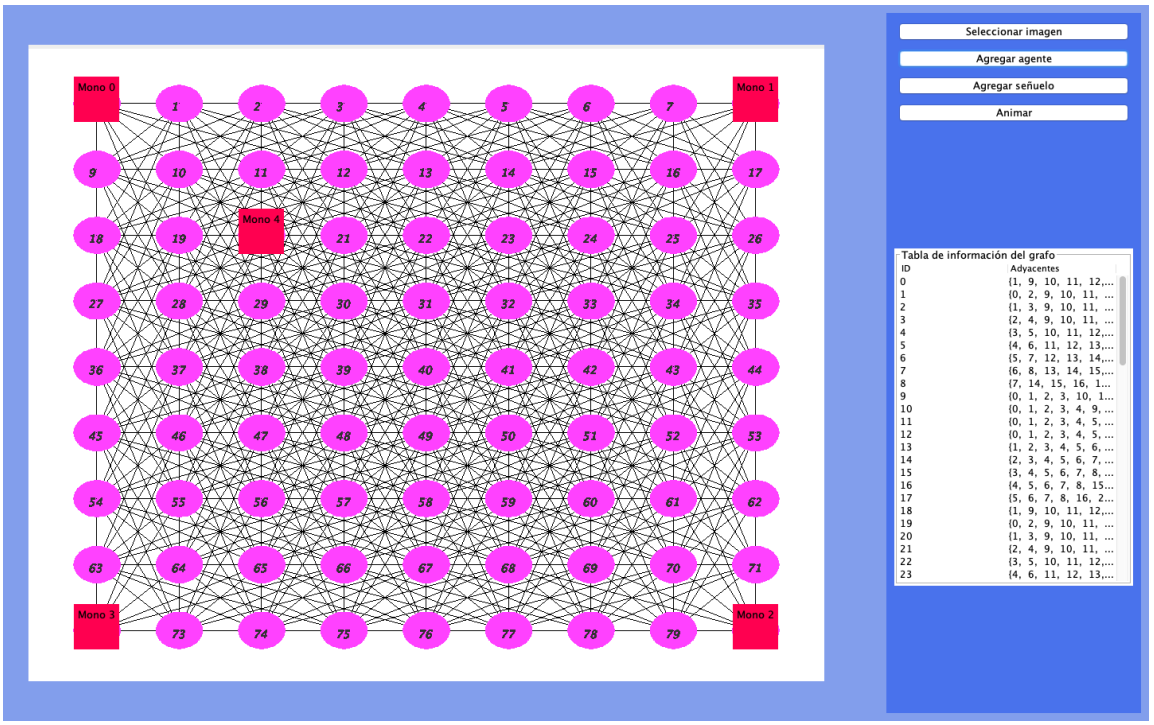
Para esta actividad hice uso de mis conocimientos en grafos para calcular como recuperar el movimiento de cada uno de los agentes mientras hacen su búsqueda para llegar el señuelo.

4.1 Calculo del posicionamiento de cada uno de los agentes o del señuelo

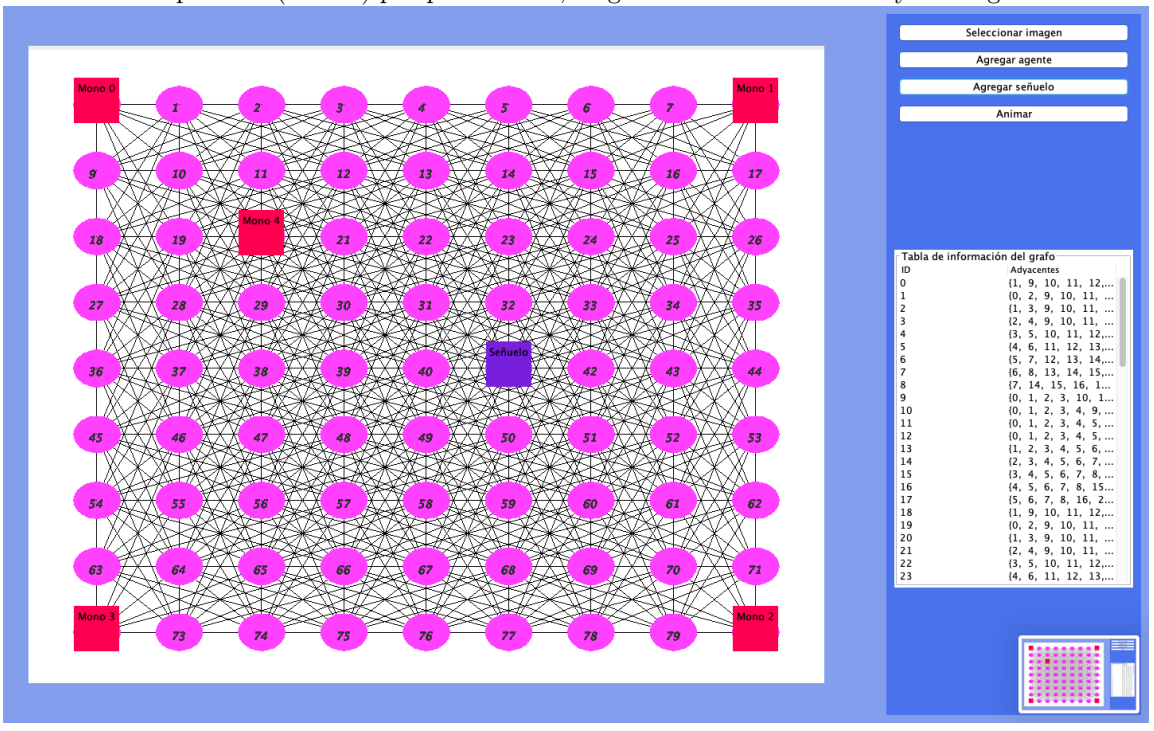
1. Para hacer esto, primero reduci la imagen a una dimensión de 1000 x 800, después de eso la coloqué en un JPanel y mediante un listener obtengo la coordenada correcta, pero para tener más precisión busco cuál de los vértices es el más cercano a él.



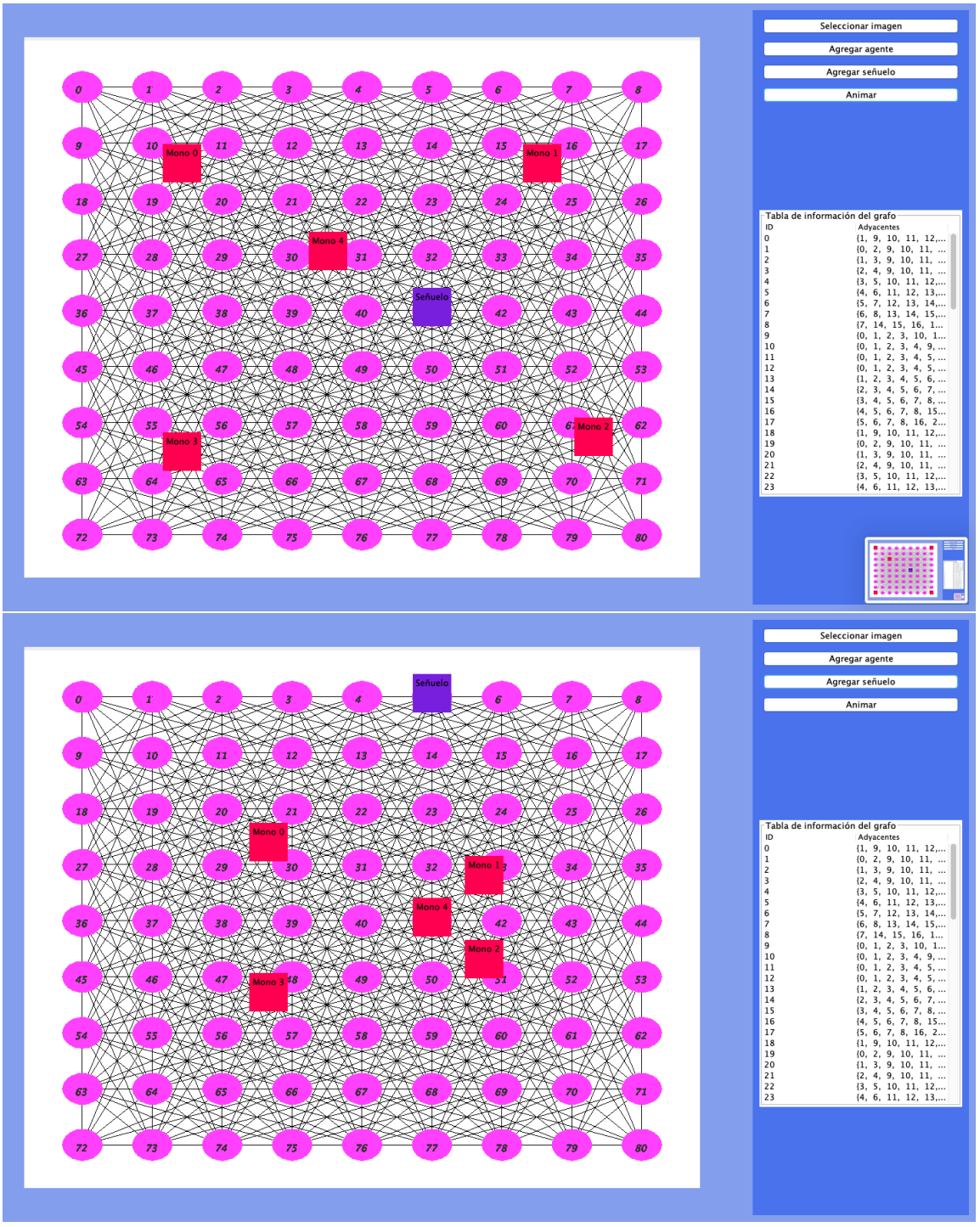
2. Para agregar un agente es necesario dar en "Agregar agente" y posterior a eso puedes agregar hasta la cantidad de vértices existentes - 1, ya que necesitas agregar siempre un señuelo

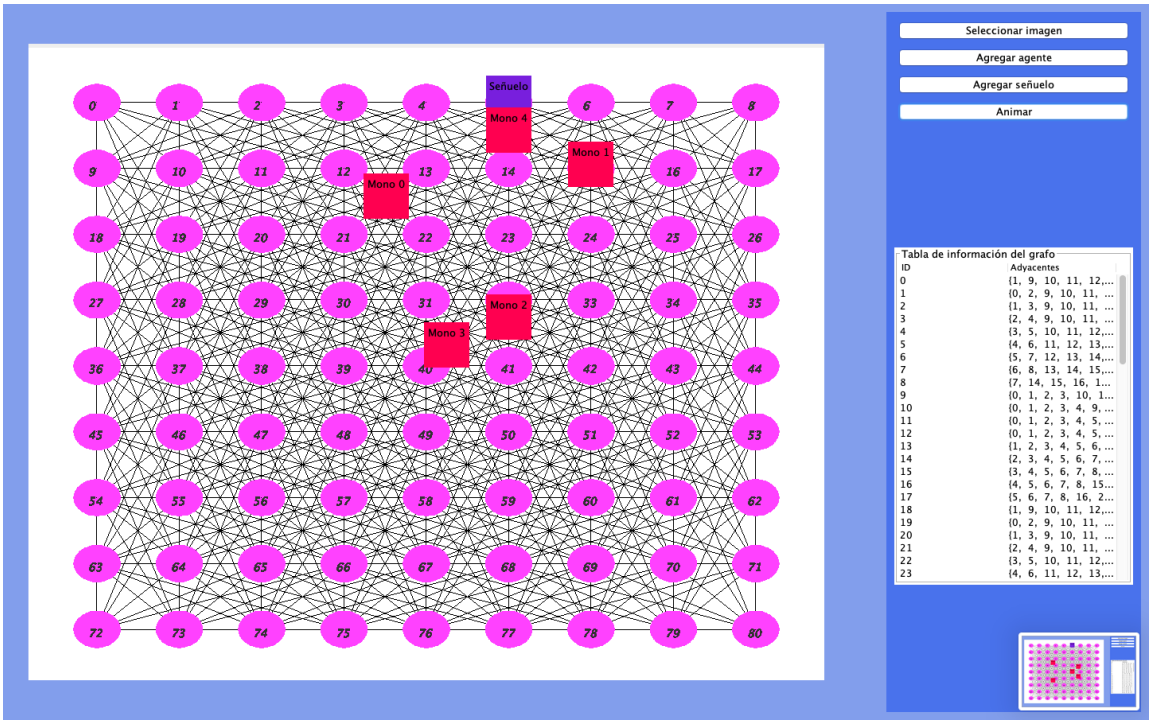


3. Para agregar un señuelo presionamos el botón "Agregar señuelo", tomando en cuenta que sólo se puede colocar en una posición (vértice) por primera vez, luego de esto será definitiva y lo obligará a animar.



4. Finalmente para animar habrá que dar en el botón animar que se encargará de mover a los agentes mientras ninguno haya llegado al señuelo, después de eso el proceso acabará y todos dejarán de moverse.




Señuelo
Mono 4
Mono 1
Mono 0
Mono 2
Mono 3
Seleccionar imagen
Agregar agente
Agregar señuelo
Animar
Tabla de información del grafo
ID
Adyacentes
0 {1, 9, 10, 11, 12,...
1 {0, 2, 9, 10, 11, ...
2 {1, 3, 9, 10, 11, ...
3 {2, 4, 9, 10, 11, ...
4 {3, 5, 10, 11, 12,...
5 {4, 6, 11, 12, 13,...
6 {5, 7, 12, 13, 14,...
7 {6, 8, 13, 14, 15,...
8 {7, 14, 15, 16, 1...
9 {0, 1, 2, 3, 10, 1...
10 {0, 1, 2, 3, 4, 9, ...
11 {0, 1, 2, 3, 4, 5, ...
12 {0, 1, 2, 3, 4, 5, ...
13 {1, 2, 3, 4, 5, 6, ...
14 {2, 3, 4, 5, 6, 7, ...
15 {3, 4, 5, 6, 7, 8, ...
16 {4, 5, 6, 7, 8, 15...
17 {5, 6, 7, 8, 16, 2...
18 {1, 9, 10, 11, 12,...
19 {0, 2, 9, 10, 11, ...
20 {1, 3, 9, 10, 11, ...
21 {2, 4, 9, 10, 11, ...
22 {3, 5, 10, 11, 12,...
23 {4, 6, 11, 12, 13,...

5 Códigos utilizados

Se utilizó el siguiente código para "animar" a los agentes con el hilo.

```
%public void run(){
  try{
      while( haySenuelo ){
          for(int i = 0; i < movimientos; i++){
              panelPorCapasImagenFondo.add(senuelo, 0);
              for(Agente agente: agentes){
                  agente.avanza();
                  panelPorCapasImagenFondo.add(agente, 0);
              }
              Thread.currentThread().sleep(tiempoDeEspera);

              synchronized(this){
                  while(detener == true){
                      wait();
                  }
              }
          }
          haySenuelo = false;
          senuelo.setVisible(false);
          panelPorCapasImagenFondo.add(senuelo, 0);
          senuelo.existe = false;
          for(Agente agente: agentes) {
              if (agente.pos == senuelo.pos) {
                  agente.velocidad += 2;
                  break;
              }
          }
          detente();
      }
  }catch(Exception e){
      e.printStackTrace();
  }
}
```

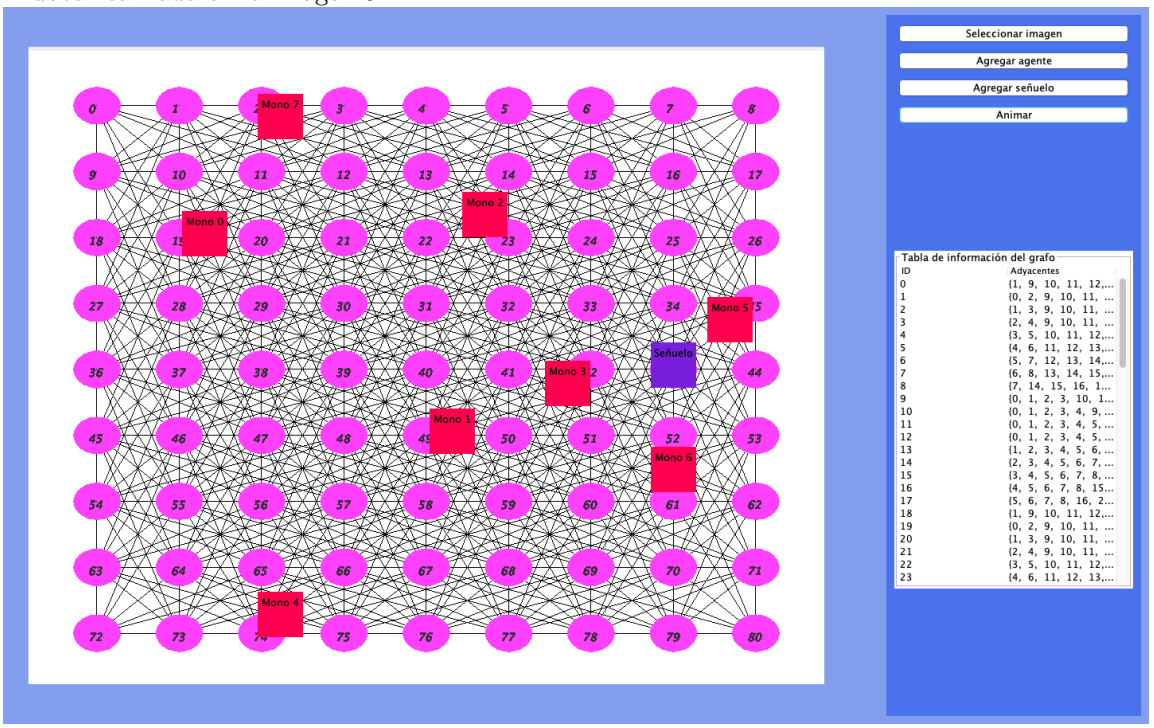
Se utilizó el siguiente código para "calcular" cómo moverse de un nodo hacía todos los siguientes mediantes el algoritmio de A*

```
%public double h(Arista ari) {
    return Math.hypot(ari.a.cir.centro.x - ari.b.cir.centro.x,
       ari.a.cir.centro.y - ari.b.cir.centro.y);
}

public double g(Arista ari) {
    return ari.linea.size();
}
```

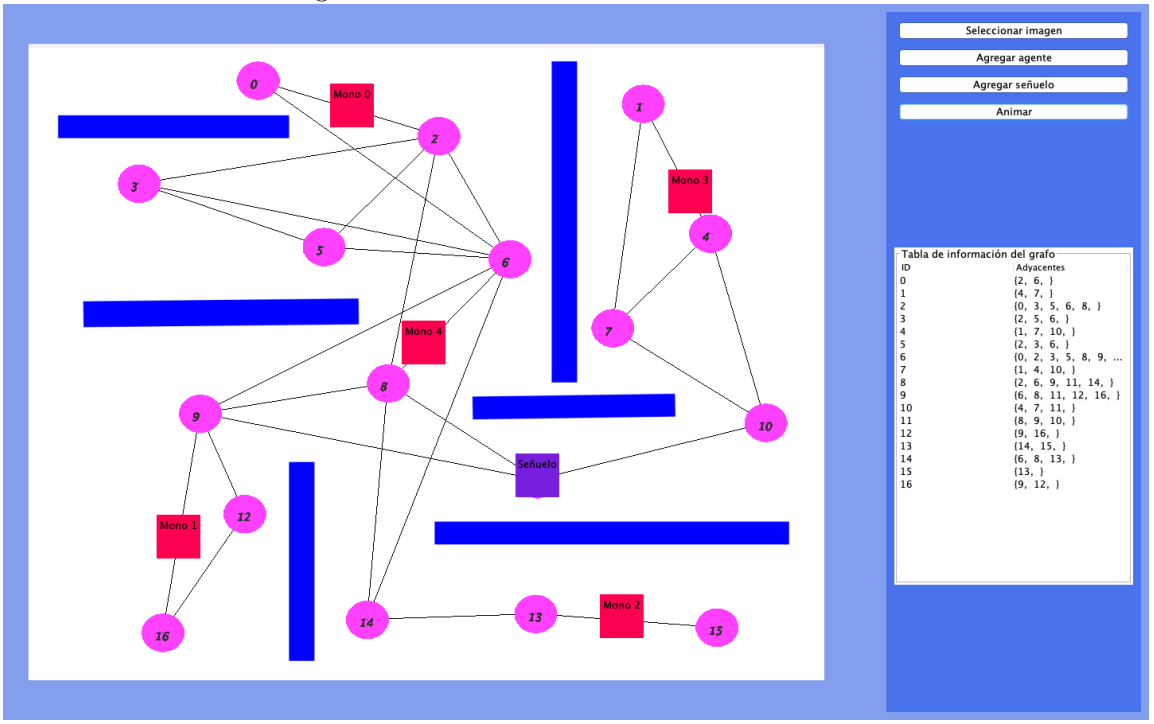
```
public void a(int s, int[] camino, int[] usando){
    double[] menosDistancia = new double[grafo.vertices.size()];
    for(int i = 0; i < menosDistancia.length; i++) {
        menosDistancia[i] = Double.MAX_VALUE / 4;
        camino[i] = -1;
    }
    PriorityQueue<Estado> pq = new PriorityQueue<Estado>(20, new
       Comparator<Estado>() {
        @Override
        public int compare(Estado o1, Estado o2) {
            return o1.distancia < o2.distancia ? -1: 1;
        }
    });

    Estado inicio = new Estado(s, 0);
    pq.add(inicio);
    menosDistancia[s] = 0;
    while( !pq.isEmpty() ){
        Estado anterior = pq.poll();
        if( anterior.distancia != menosDistancia[anterior.u] ){
            continue;
        }
        for(int id: grafo.ady.get(anterior.u)) {
            Arista ari = grafo.aristas.get(id);
            Estado actual = new Estado(ari.b.id, anterior.distancia +
               h(ari) + g(ari));
            if( actual.distancia < menosDistancia[actual.u] ) {
                menosDistancia[actual.u] = actual.distancia;
                camino[actual.u] = anterior.u;
                usando[actual.u] = id;
                pq.add(actual);
            }
        }
    }
}
```

6 Pruebas y resultados

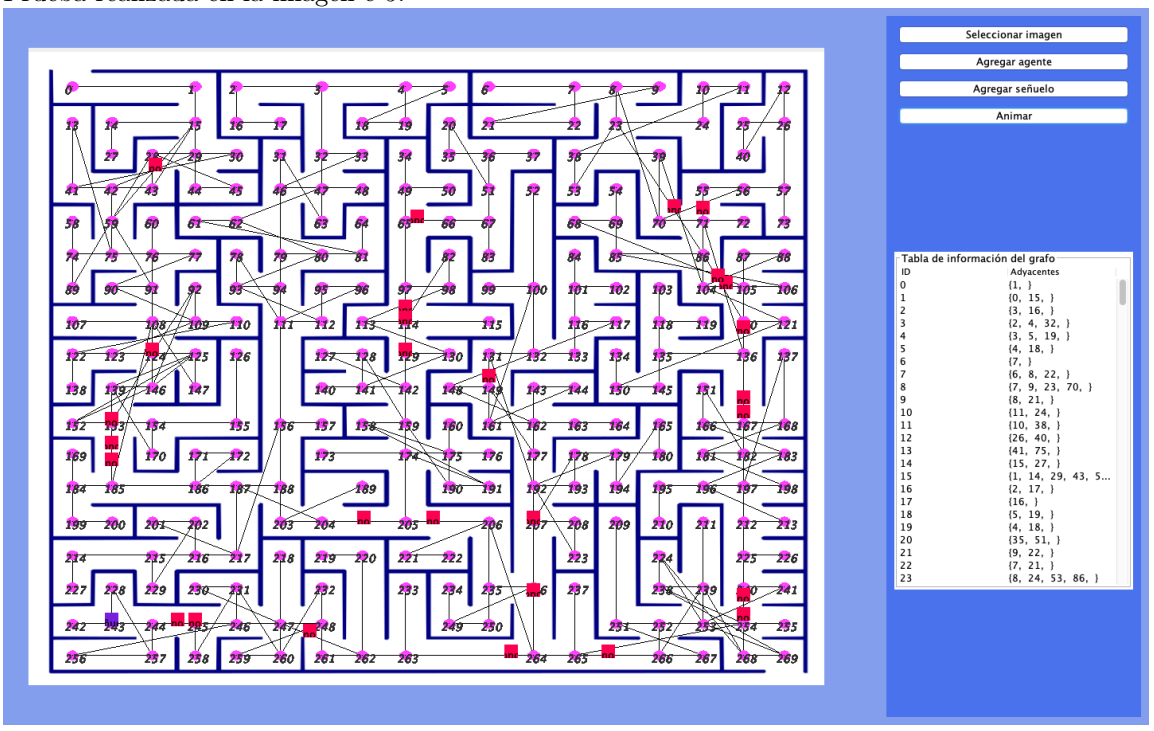
1. Prueba realizada en la imagen 3-1.



2. Prueba realizada en la imagen 3-4.



3. Prueba realizada en la imagen e-5.



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.
Además de pensar cómo representar el grafo de tal manera que se evitara el calculo de ciertas cosas varias veces (el calculo de las líneas, de si existe un arista, ...)
Además es bueno pensar en cómo se puede simular el movimiento del objeto para que sea lo más natural posible, que no se vea que se está teletransportando, para esto guarde la arista por puntos entonces el movimiento lo hago cada k-puntos de la arista y simulo un proceso más natural.

8 Apéndice(s)

https://medium.com/@nicholas.w.swift/easy-a-star-pathfinding-7e6689c7f7b2 (Algoritmo A*)
https://www.geeksforgeeks.org/bresenhams-line-generation-algorithm/ (Algoritmo de Bresenham)
https://docs.oracle.com/javase/tutorial/uiswing/events/mouselistener.html (Selección del pixel)
https://www.hackerearth.com/practice/notes/graph-theory-part-i/ (Grafos)