



UNIVERSIDAD DE  
GUADALAJARA

CENTRO UNIVERSITARIO DE CIENCIAS  
EXACTAS E INGENIERIAS

SEMINARIO DE SOLUCIÓN A PROBLEMAS DE ALGORITMIA

---

## Actividad 5: Dijkstra

---

*Presentado a:*

David Alejandro Gómez Anaya

Departamento de Ciencias Computacionales

*Presentado por:*

Abraham Baltazar Murillo Sandoval, 218744368

Seccion D15, 2019 B

## 1 Diagrama de clases propuesto

Para esta actividad no necesite hacer una nueva clase ya que las demás clases que tenía cumplían el cometido deseado

## 2 Objetivo

Diseñar un sistema en base el sistema anterior capaz de mover un objeto entre vértice origen y destino utilizando el camino menos costoso, es decir, a través de Dijkstra mover al agente. Además debe de ser capaz de mostrar el camino mínimo en caso de existir desde el vértice en donde se encuentra el agente a un vértice x seleccionado por el usuario desde pantalla.

## 3 Marco teorico

### 3.1 Recuperacion del camino

Como la dijkstra es expansivo, y va nivel por nivel, entonces a cada vértice le podemos asignar un padre. Entonces para recuperar el camino sólo necesitamos desde el vértice en el que estamos subir vértice por vértice siguiendo los padres, hasta que llegamos al vértice origen donde se colocó el agente.

### 3.2 Dijkstra

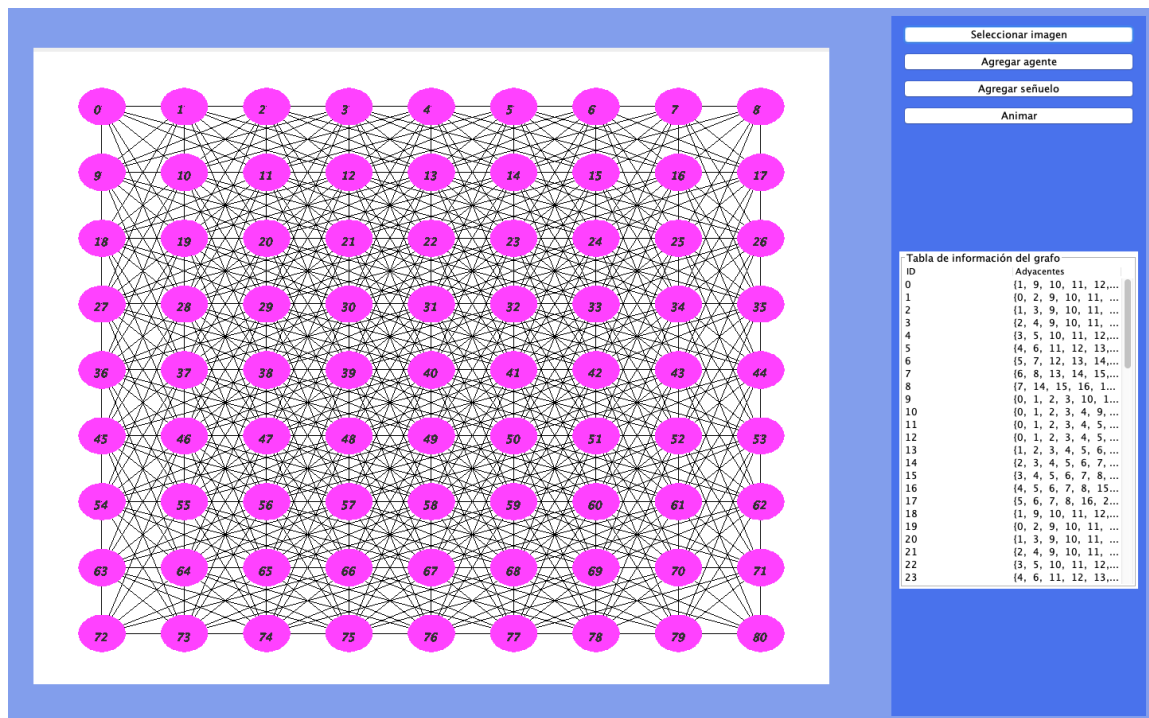
El algoritmo de Dijkstra es un algoritmo para encontrar el camino más corto entre dos vértice en un grafo, que pueden representar por ejemplo, las distancias entre una red. Fue inventado por el científico Edsger Dijkstra en 1956 y publicado 3 años después.

El algoritmo existe en muchas variantes. El algoritmo original de Dijkstra encontraba la distancia más corta entre dos vértice, pero con una pequeña modifica cambiando a un vértice como el origen se puede calcular la distancia más corta desde ese vértice a todos los demás vértices en caso de ser conexos, formando el árbol de distancias mínimas.

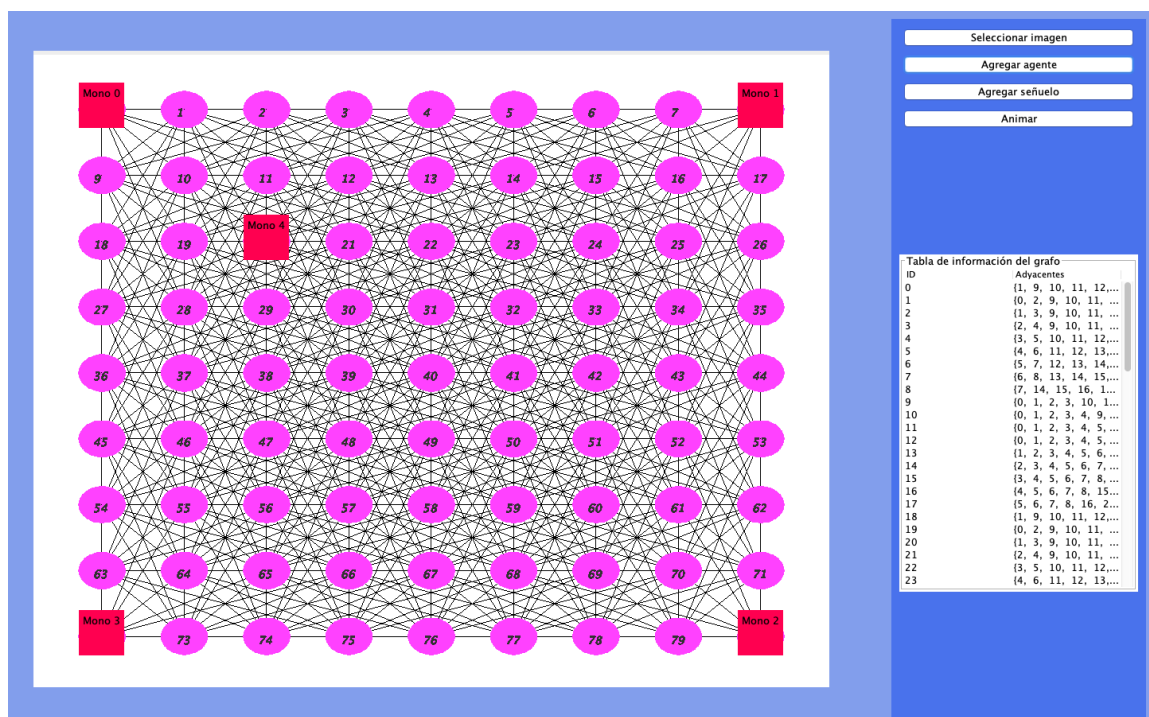
## 4 Desarrollo

### 4.1 Calculo del posicionamiento de cada uno de los agentes o del senuelo

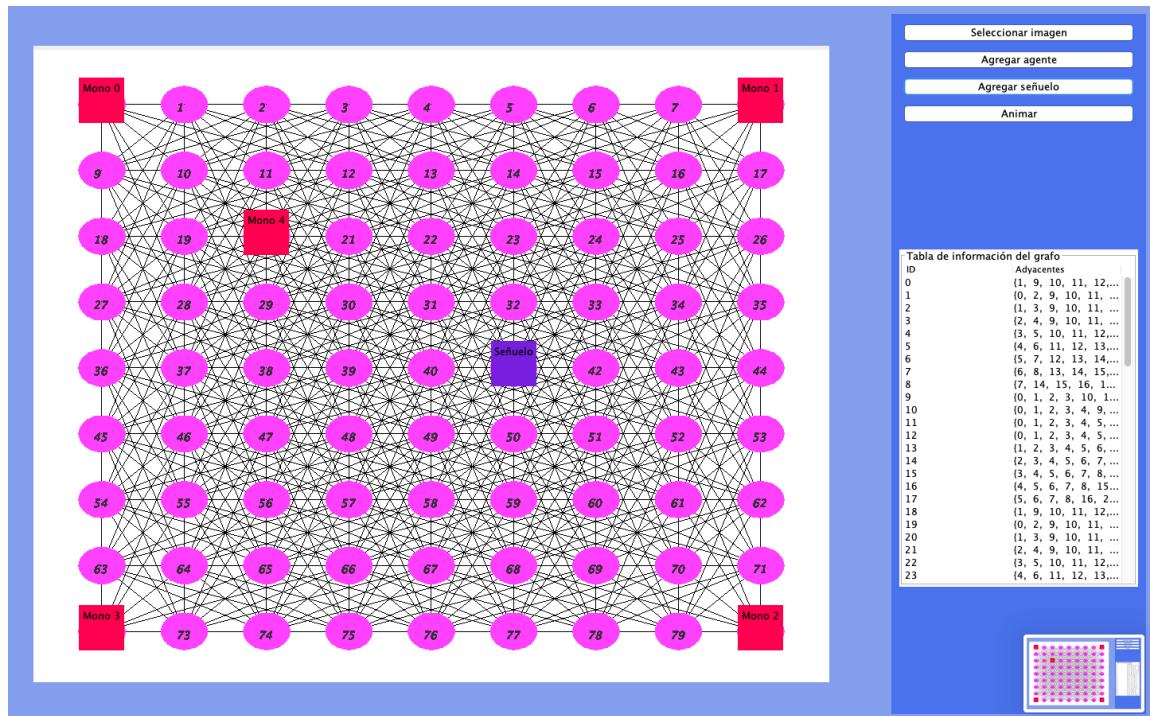
1. Para hacer esto, primero reduje la imagen a una dimensión de 1000 x 800, después de eso la coloqué en un JPanel y mediante un listener obtengo la coordenada correcta, pero para tener más precisión busco cuál de los vértices es el más cercano a él.



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.

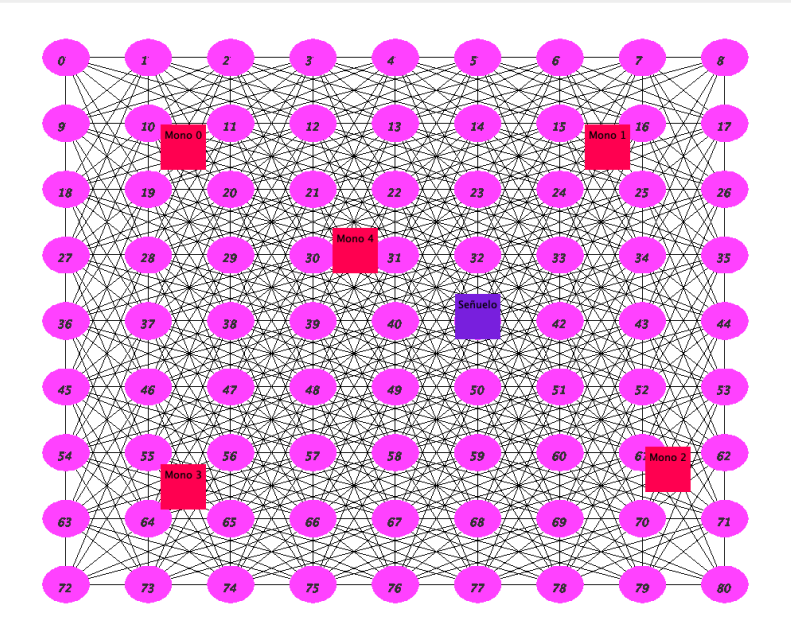
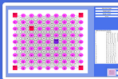


Tabla de información del grafo

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,...



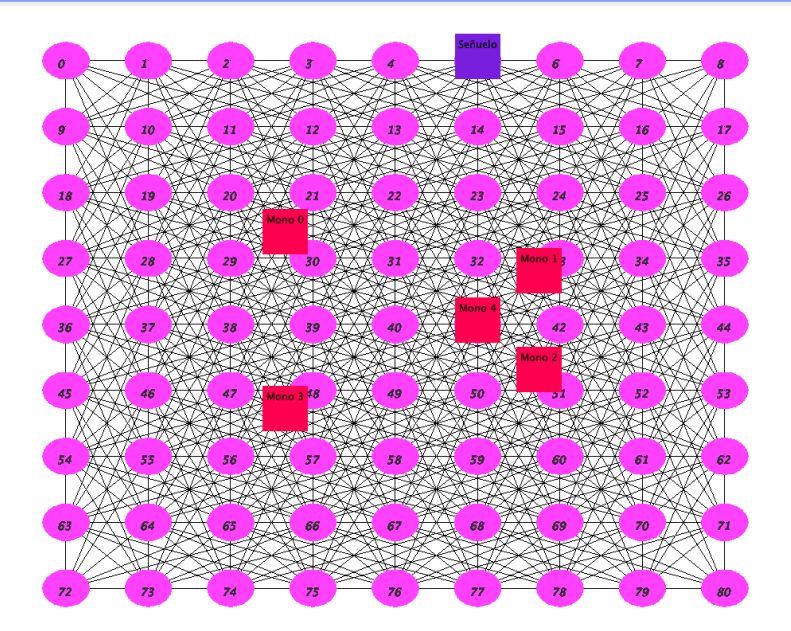
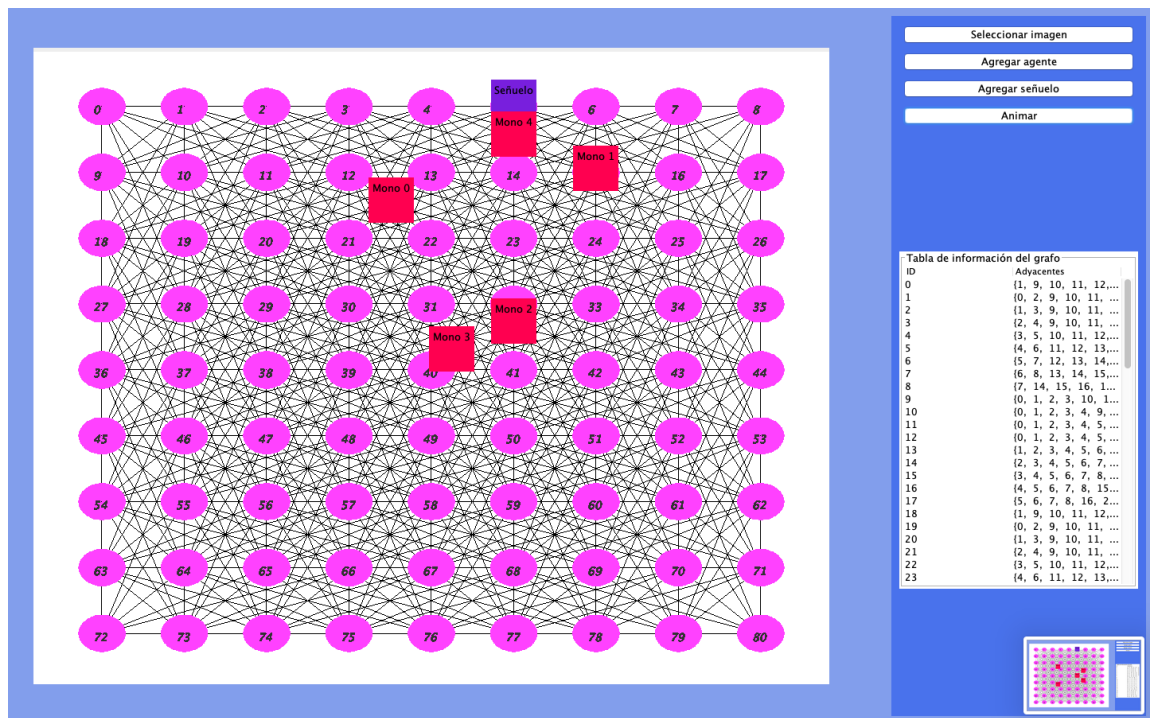


Tabla de información del grafo

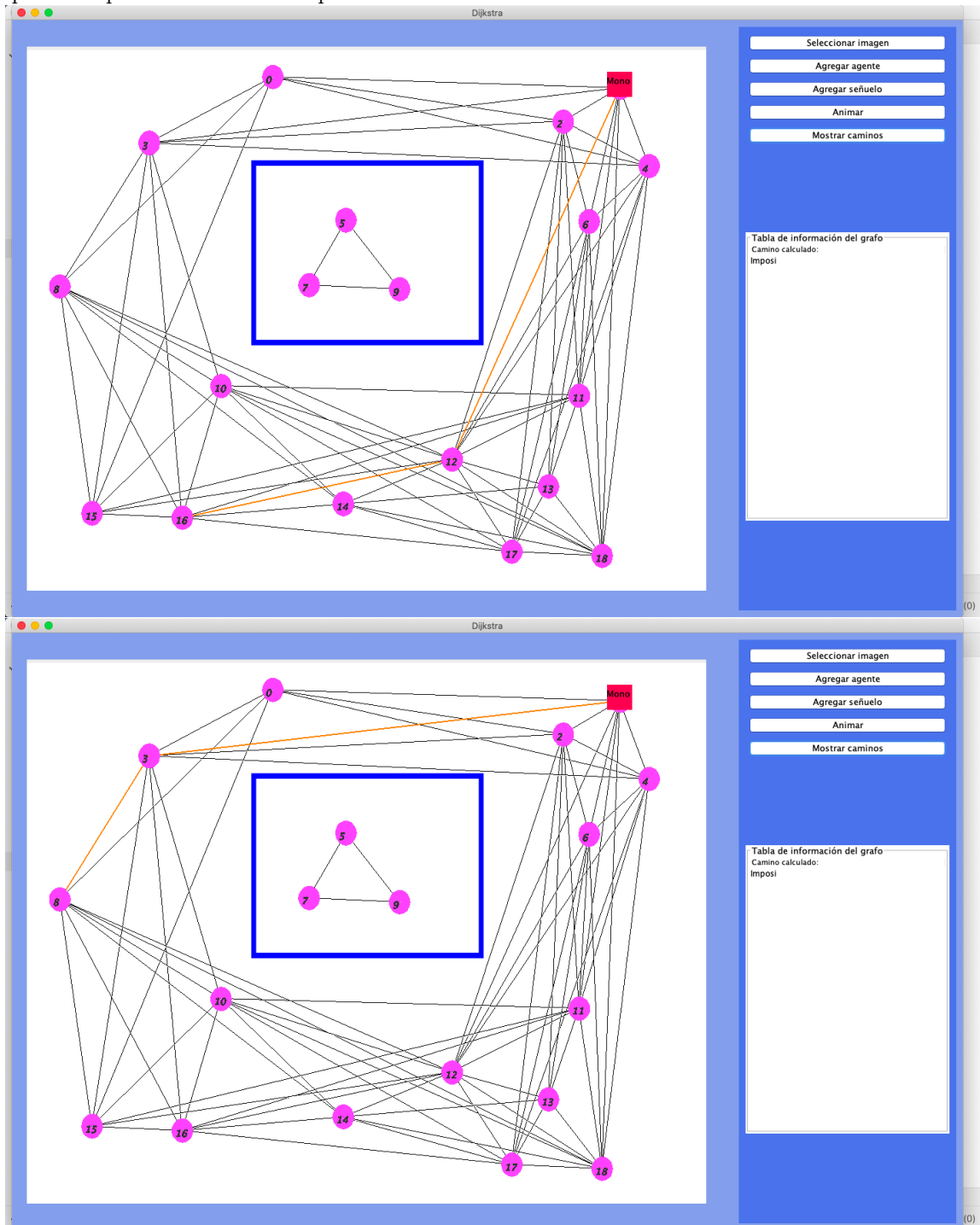
ID	Adyacentes
0	(1, 9, 10, 11, 12,...
1	(0, 2, 9, 10, 11, ...
2	(1, 3, 9, 10, 11, ...
3	(2, 4, 9, 10, 11, ...
4	(3, 5, 10, 11, 12,...
5	(4, 6, 11, 12, 13,...
6	(5, 7, 12, 13, 14,...
7	(6, 8, 13, 14, 15,...
8	(7, 14, 15, 16, 1,...
9	(0, 1, 2, 3, 10, 1,...
10	(0, 1, 2, 3, 4, 9,...
11	(0, 1, 2, 3, 4, 5,...
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,...





## 4.2 Cálculo de camino a seguir para llegar a un vértice x

1. Para hacer esto desde el vértice en donde se encuentra el agente, empiezo a hacer un Dijkstra, el cual para cada vértice guarda un padre. Cuando llego al vértice destino entonces lo que hago es seguir la ruta padre  $\rightarrow$  padre(padre) ... hasta llegar al vértice origen y es así donde pinto las aristas que corresponden al camino empleado.



## 5 Codigos 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();
    }
}
```

Código de Dijkstra empleado para calcular caminos mínimos

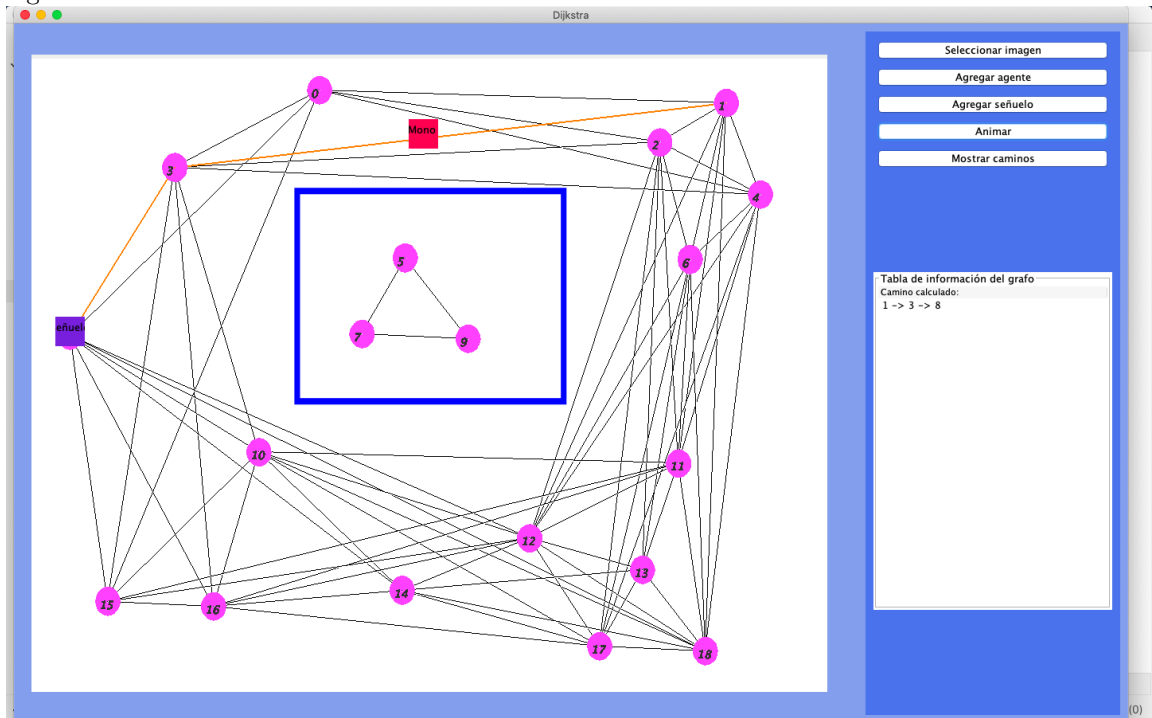
```
%public void dijkstra(int s, int[] camino, int[] usando){
    double[] menosDistancia = new double[grafo.vertices.size()];
    for(int i = 0; i < menosDistancia.length; i++) {
        menosDistancia[i] = Double.MAX_VALUE / 4;
        camino[i] = -1;
        usando[i] = -1;
    }
    PriorityQueue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



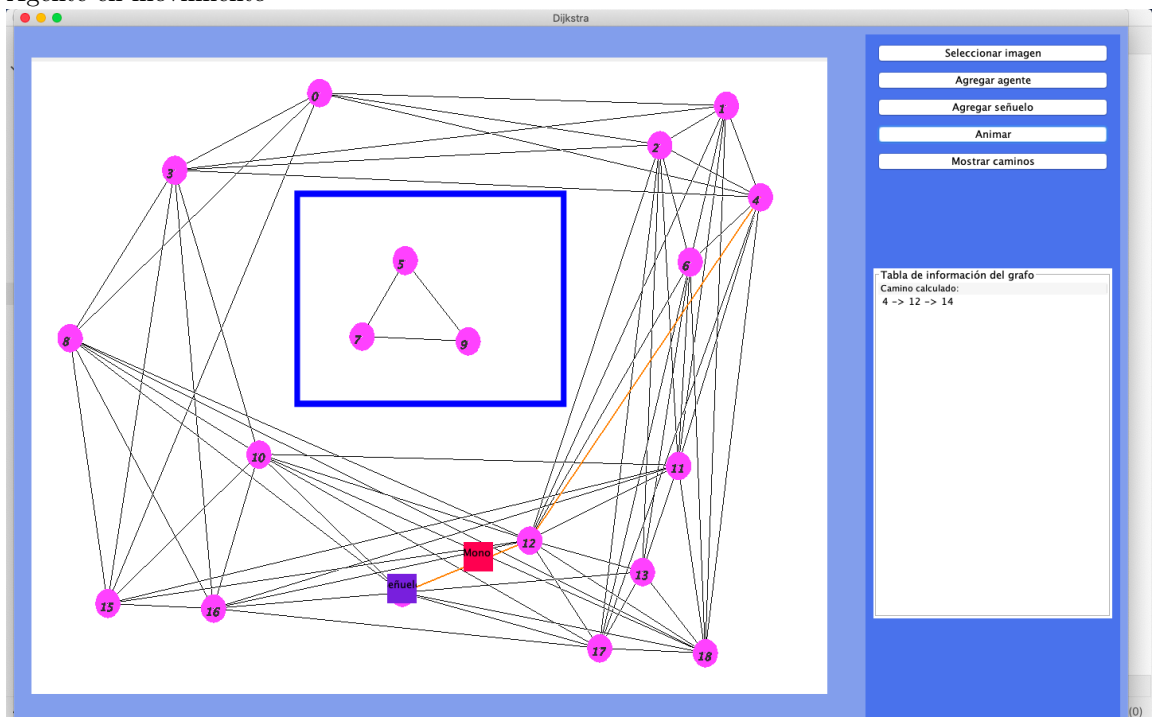
```
        public int compare(Estado o1, Estado o2) {
            return o1.distancia < o2.distancia ? -1: 1;
        }
    });
    Estado inicio = new Estado(s, 0);
    pq.add(inicio);
    menosDistancia[s] = 0;
    while( !pq.isEmpty() ){
        Estado anterior = pq.poll();
        if( anterior.distancia != 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 +
                ari.a.cir.distancia(ari.b.cir));
            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. Agente en movimiento



### 2. Agente en movimiento



## 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, ...)

## 8 Apendices

<https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/> (Dijkstra)

<https://www.geeksforgeeks.org/lowest-common-ancestor-binary-tree-set-1/> (LCA)