More Algorithms and Data Structures

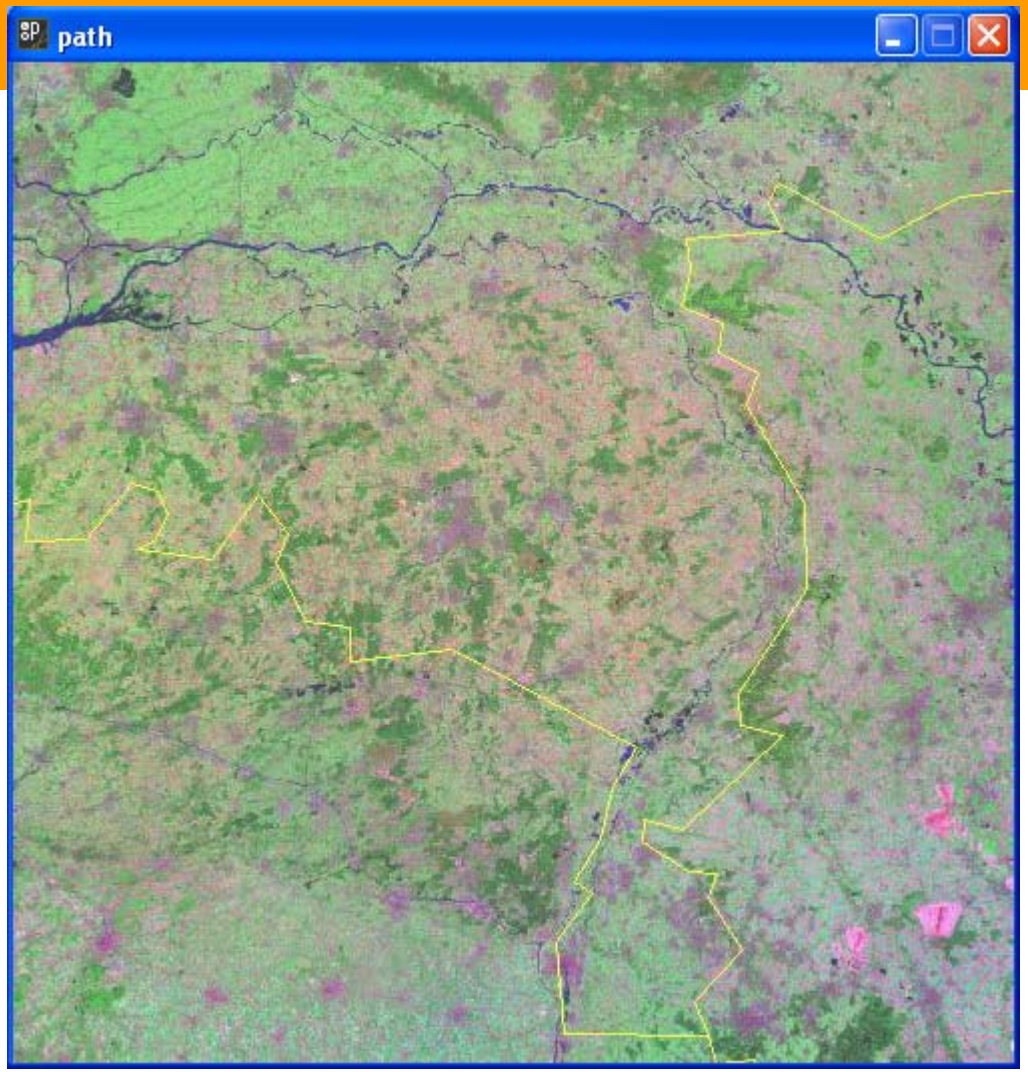
DG200 team (contribution Loe Feijs)

TU/e Technische Universiteit Eindhoven University of Technology

Where innovation starts

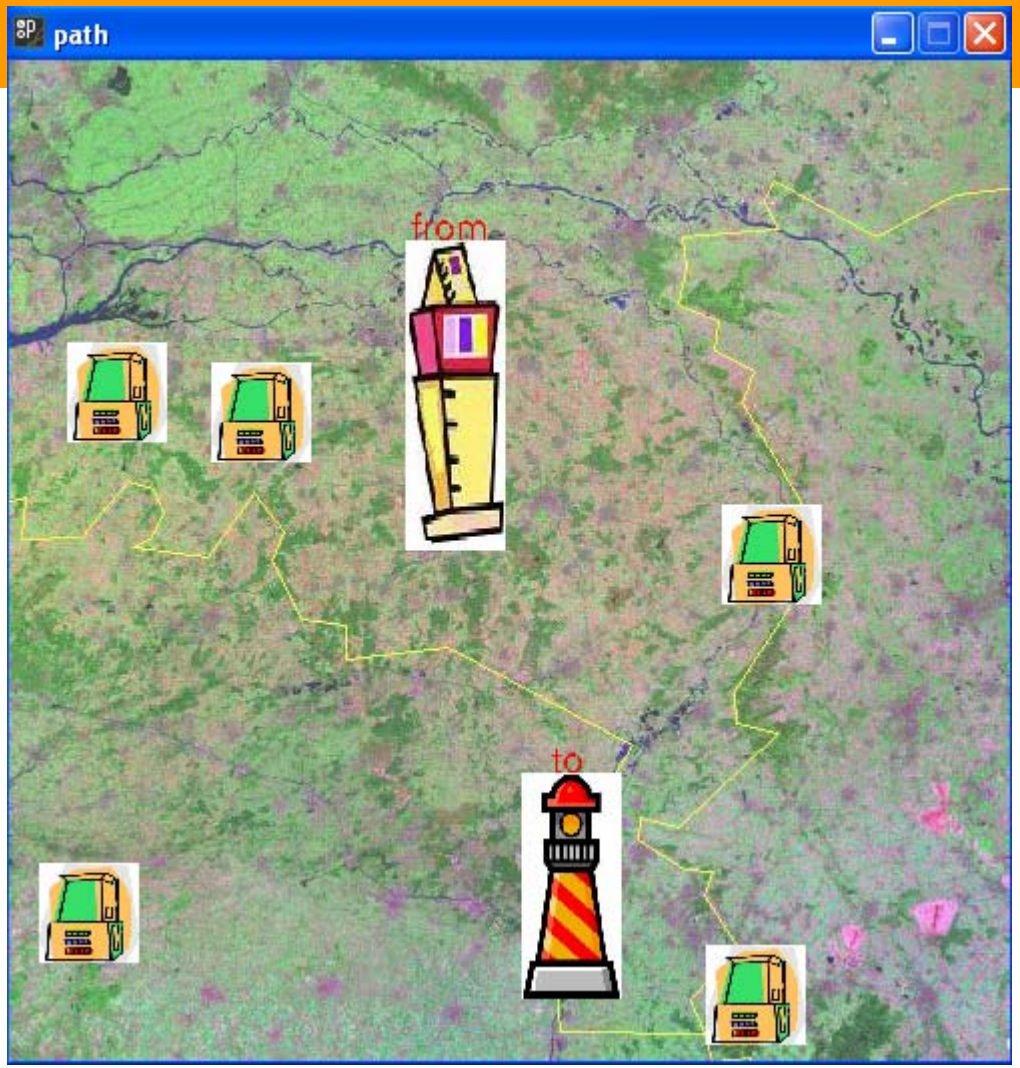
Optimising

- Demo application
  - edit a landscape with one-tower cities
  - three tower types
  - choose levels
  - random graph
  - shortest path algorithm
  - read: wikipedia, Dijkstra’s algorithm


path

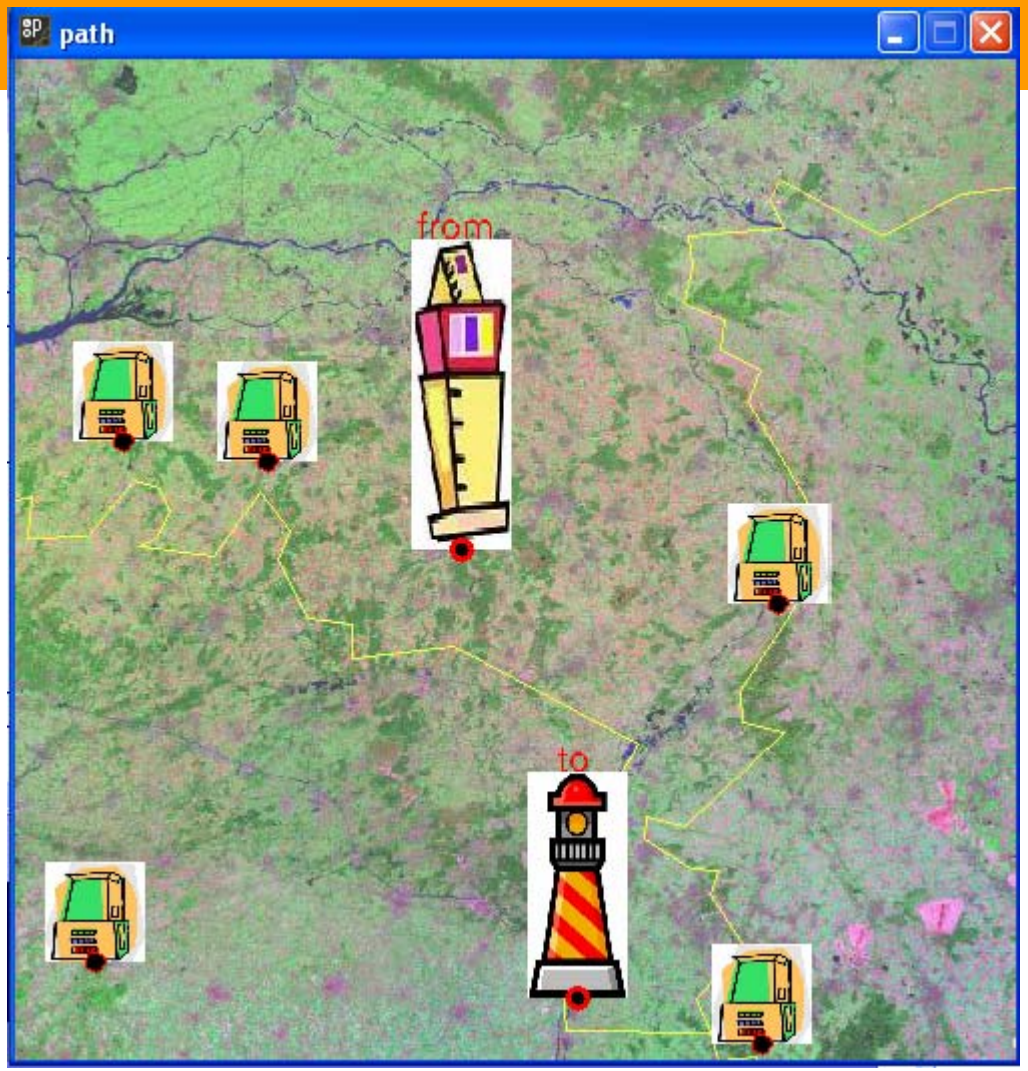
mouse

arrow keys



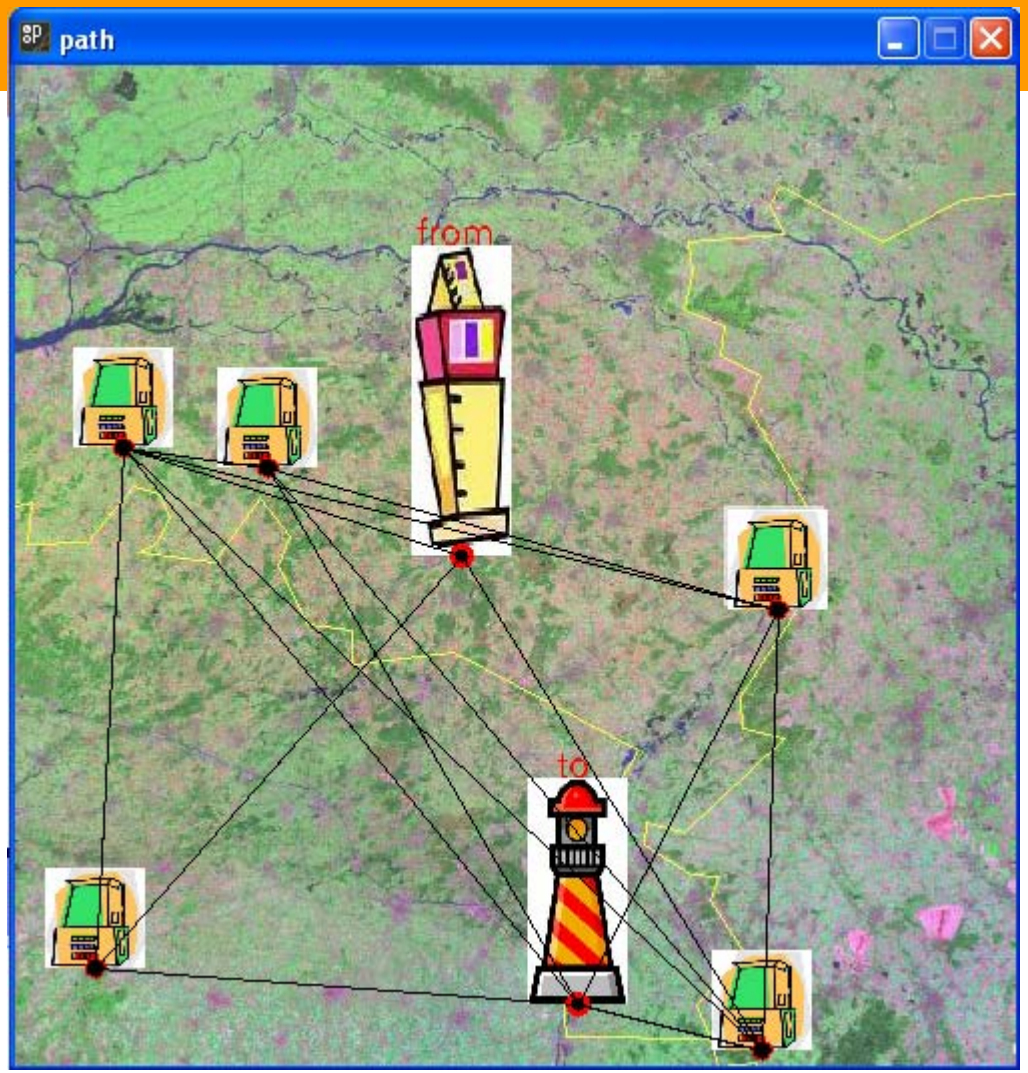
enter

```
showNodes();
```



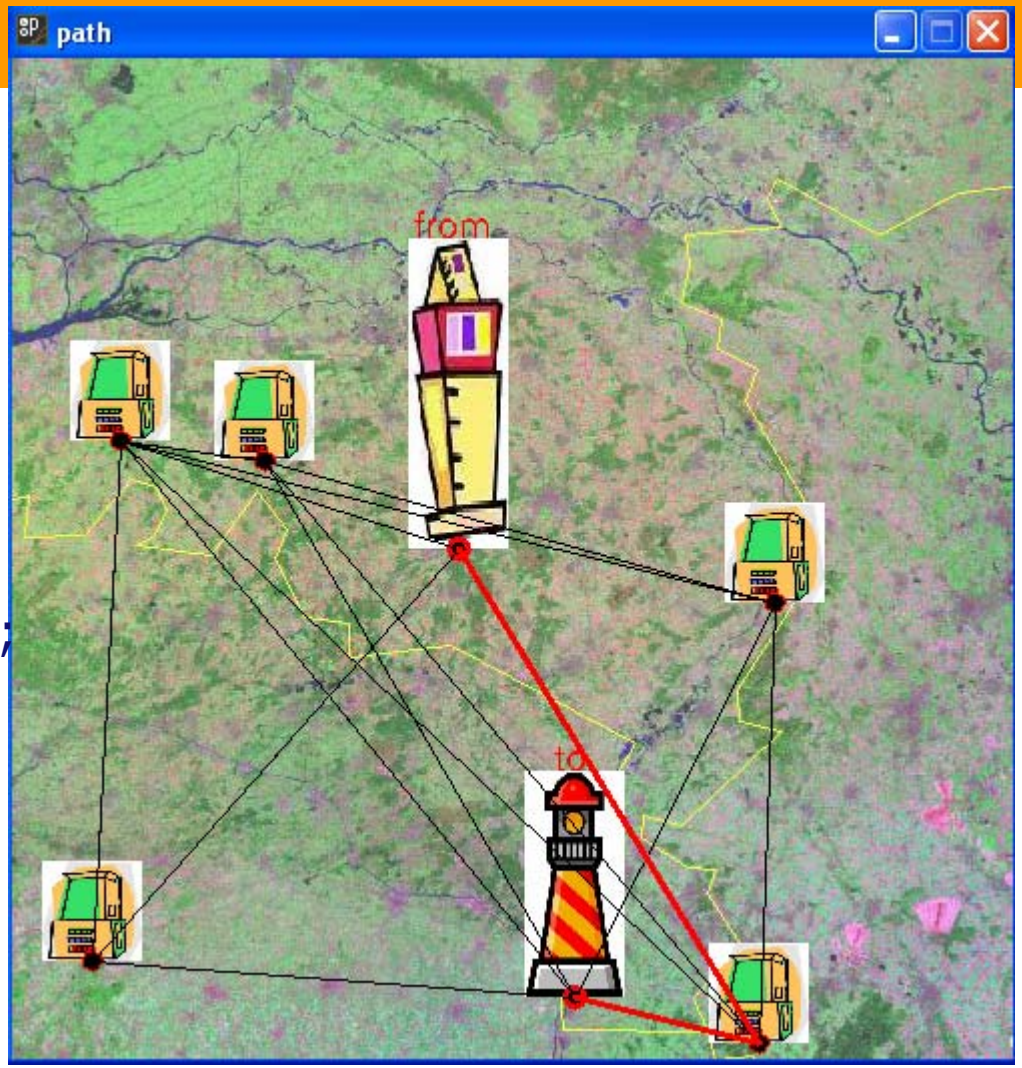
enter

```
makeGraph();
showEdges();
```



enter

```
prvs=dijkstra();
showpath(prvs);
```



Optimising

- Dijkstra's shortest path algorithm
  - invented in 1959
  - wikipedia, Dijkstra path
  - to be translated into processing

```
 1  function Dijkstra(Graph, source):
 2      for each vertex v in Graph:             // Initializations
 3          dist[v] := infinity                 // Unknown distance function from source to v
 4          previous[v] := undefined            // Previous node in optimal path from source
 5      dist[source] := 0                       // Distance from source to source
 6      Q := the set of all nodes in Graph      // All nodes in the graph are unoptimized
 7      while Q is not empty:                   // The main loop                  - thus are in Q
 8          u := node in Q with smallest dist[]
 9          remove u from Q
10          for each neighbor v of u:           // where v has not yet been removed from Q.
11              alt := dist[u] + dist_between(u, v)
12              if alt < dist[v]                // Relax (u,v)
13                  dist[v] := alt
14                  previous[v] := u
15      return previous[]
```

Optimising

```
int[] dijkstra() {
// see http://en.wikipedia.org/wiki/Dijkstra_algorithm
// takes distBetween as graph and index zero as source
   int undefined = -1;
   dIst = new int[towers];
   int[] previous = new int[towers];
   for (int v=0; v<towers; v++) {        // Initializations
       dIst[v] = infinity;               // Unknown distance function from source to v
       previous[v] = undefined;          // Previous node in optimal path from source
   }
   int source=0;
   dIst[source] = 0;                     // Distance from source to source
   boolean[] Q = new boolean[towers];    // All nodes in the graph are unoptimized -
   for (int i=0;i<towers;i++) {
       Q[i]=true;                        // thus are in Q
   }
   while (isNotEmpty(Q)) {               // The main loop
         int u = nodeWithSmallestDistIn(Q);
         Q[u]=false;
         for (int v=0; v<towers; v++) {  // where v has not yet been removed from Q.
             if (v!=u && distBetween[u][v] < infinity && Q[v]) {
                int alt = dIst[u] + distBetween[u][v];
                if (alt < dIst[v]) {     // Relax (u,v)
                   dIst[v] = alt;
                   previous[v] = u;
                }
             }
         }
    }
    return previous;
}
```

Optimising

- Auxilaries needed
  - e.g. manipulating a set of numbers
  - here coded as boolean array named q
  - non-emptiness test to be done by searching

```
boolean isNotEmpty(boolean[] q){
   int i=0;
   boolean isEmpty=true;
   while (i<towers && isEmpty) {
         if (q[i]) isEmpty=false;
         i++;
   }
   return !isEmpty;
}
```

Optimising (student work)

- Exercises
  - can you invent your own shortest path algorithm?