

# Exercise 4

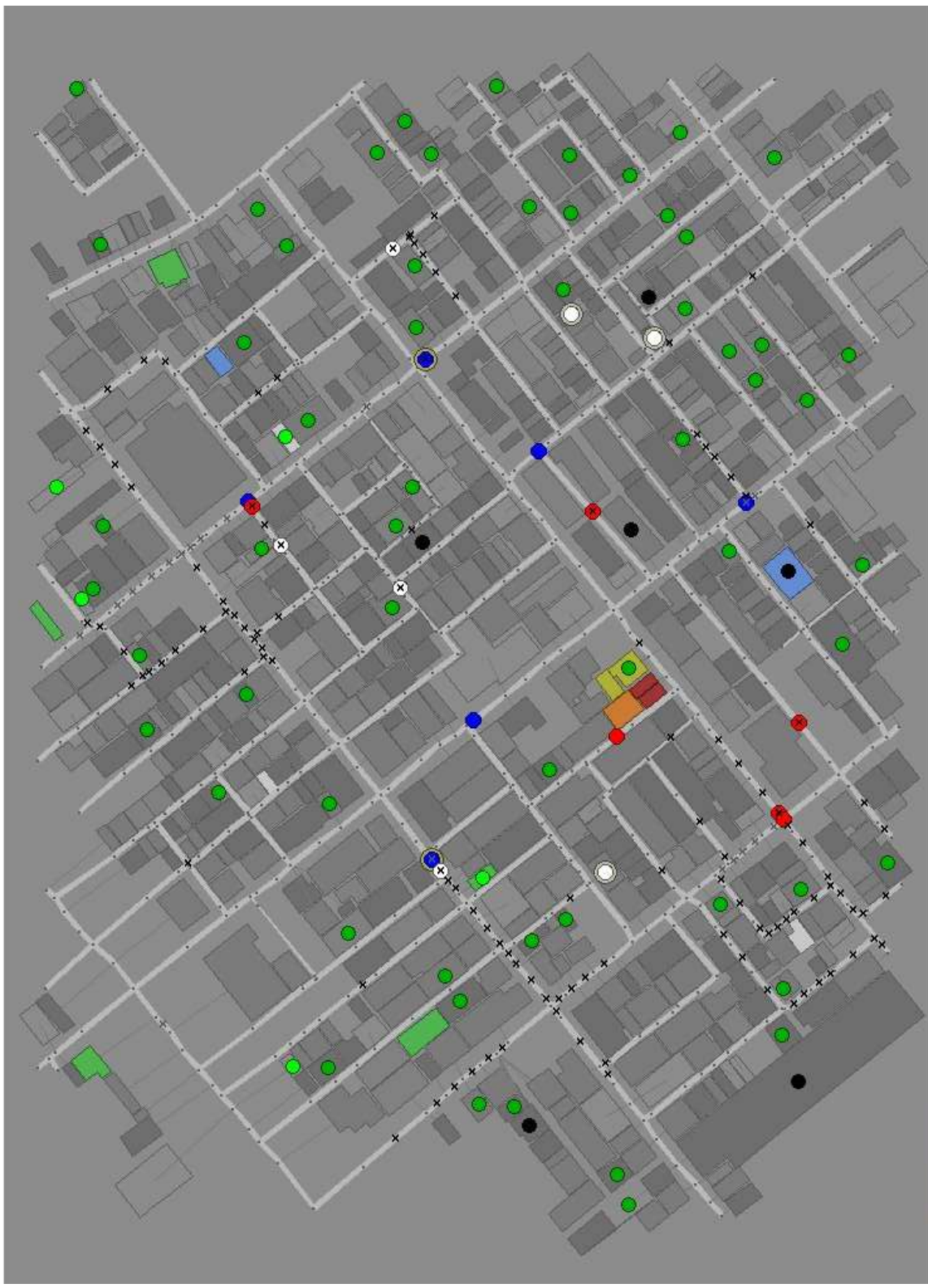
# Path planning in RCRSS

- General: Find path or costs from position A to B
- Basic: A is always agent's position
- Agent's code usually needs general solution for evaluation of different targets
- Cost query might be called a lot, path usually only once
- → Need efficient calculation: cost A to B

# Example: Ambulance agents

- Civilians are buried: Takes  $B/N$  rounds to unbury with  $N$  ambulance teams
- Civilians are hurt: Will die, if not after  $X$  rounds at refuge
- Ex: civilian:  $B = 30$  will probably die after 15 rounds, 3 ambulance teams
- Can you save him?
- Or better save two other civilians?

Time: 54    Score: 93,647112



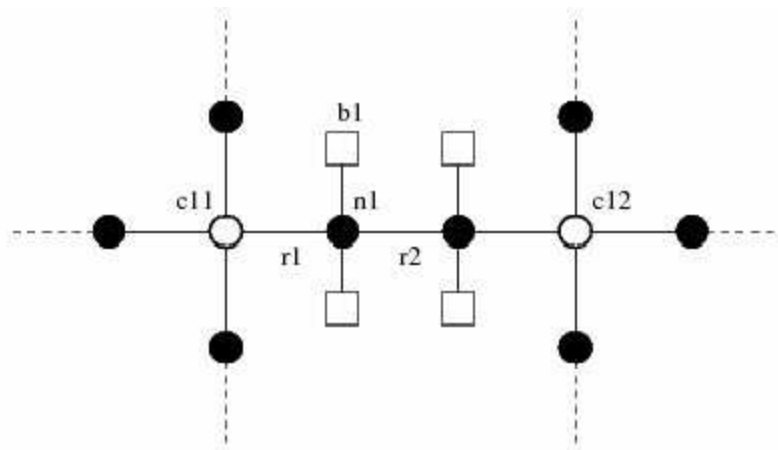
Interval  Animation

# Search space

- Search/path planning:  $G=(V, E)$  and  $c:E\rightarrow\mathbb{R}^+$
- search algorithm ( $A^*$ ,  $D^*$ , dijkstra)
- Problem solved?
- What is  $G$ ,  $c$ ?
- The search space needs to be defined to solve a real problem

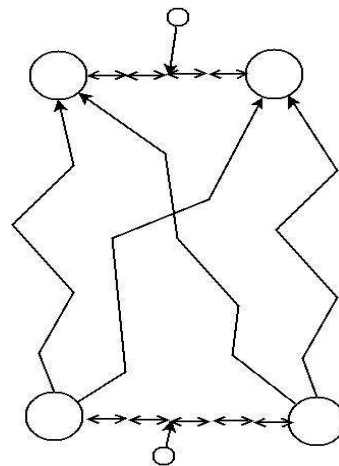
# Searching Long-roads

- Long-road nodes are nodes, that connect more than two roads (i.e. a junction)
- Efficient structuring of search space
- Users of path planning do not know about this!



# Long-roads details

- Upon a planning query you need to
  - Find the two nearest longroad nodes to start and goal (or one if start/goal is LRN)
  - Get distances to both LRNs
  - Compute minimal path for all four possibilities



# Implementation

- Keep the interface at RescueObjects, i.e. the long-roads only exist in the planner
- What you might want to do:
  - Get the long-road/long-road-nodes for a position
  - Get the „real“ roads for a long road
  - Keep the long road planner abstracted, i.e. it should really just deal with the graph
  - Get costs for a long-road

# Implementation (2)

- Calculate the longroads once and store the information for the planner
- Look in Memory class for functions how to get objects of a certain type
- Your planner should work on MotionlessObjects
- Look in SampleSearch.java as how to access objects/find neighbors, etc.
- What the simulator finally wants as a path is a list of IDs of your path

# Path Planning

- Ideally your planning nodes are designed very similar to those in the lecture
  - Parent pointer
  - Cost
  - Depth
  - Etc.
- Use predefined data structures, e.g. PriorityQueue and Set for OpenList/ClosedList
- When extracting the path: Remember you start at the goal and follow the parent
- Implement at least A\*/Dijkstra. More advanced algorithms as D\*/D\* lite, Real time A\* etc. are also fine!

# Multi-Query Path planning

- You will use the path planner for task assignment later
- Multiple (cost) queries will be done in the same round
- Store the costs of queries in a NxN matrix, which you can access like (N = #long-road nodes):
  - `costMatrix[start][goal]`

# Modifications

- Change the path planner to use the matrix instead of  $A^*$  (or similar) for costs (i.e. planning is just matrix lookup), if there is an entry
- The matrix needs to be invalidated after a world update (for now use: every round)
- Use  $A^*$ /Dijkstra to fill the rows corresponding to the nodes nearest to the agents

# Matrix row calculations

- Run  $A^*$  with no goal (i.e. explore all)
  - Without a heuristics, this is behaving like Dijkstra (SSMG)
- The closed list now contains optimal paths to each node
- Ideally: only calculate, if information is needed

# A look at the sample search

```
• public static int[] breadthFirstSearch(RescueObject start, RescueObject goal, Memory memory) {  
•     List open = new LinkedList();  
•     Map ancestors = new HashMap();  
•     open.add(start);  
•     RescueObject next = null;  
•  
• // SEARCH  
•     if (next==null) {  
•         // No path  
•         return null;  
•     }  
•     // Walk back from goal to start  
•     RescueObject current = goal;  
•     Stack path = new Stack();  
•     do {  
•         path.push(current);  
•         current = (RescueObject)ancestors.get(current);  
•     } while (current!=start && current!=null);  
•     int[] result = new int[path.size()];  
•     for (int i=0;i<result.length;++i) {  
•         result[i] = ((RescueObject)path.pop()).getID();  
•     }  
•     return result;  
• }  
• }
```

# SampleSearch (2)

- do {
- next = (RescueObject)open.remove(0);
- RescueObject[] neighbours = memory.findNeighbours(next);
- if (neighbours==null) continue;
- for (int i=0;i<neighbours.length;++i) {
- if (neighbours[i]==null) continue;
- if (neighbours[i]==goal) {
- ancestors.put(neighbours[i],next);
- next = neighbours[i];
- break;
- }
- else {
- if (!ancestors.containsKey(neighbours[i]) && !neighbours[i].isBuilding()) {
- open.add(neighbours[i]);
- ancestors.put(neighbours[i],next);
- }
- }
- }
- } while (next != goal && next != null);