Principles of Computer Game Design and Implementation

Lecture 27

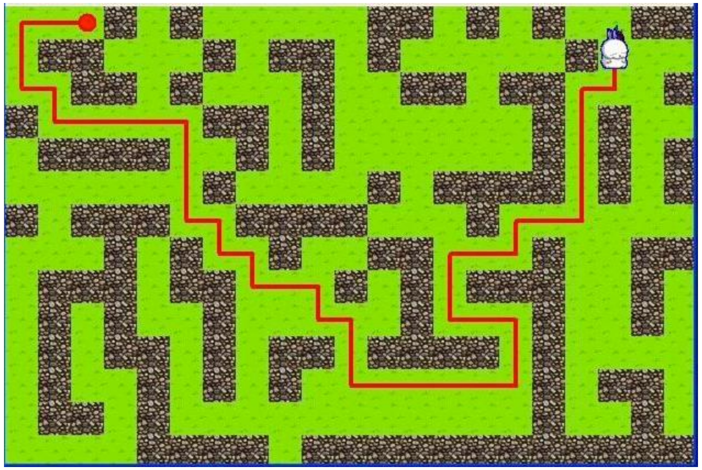
Outline for today

- pathfinding

The Problem

Pathfinding

- Given the current position and the target position
  - Calculate a sequence of positions (path)
    - Can follow with steering
    - Shortest / lowest cost path



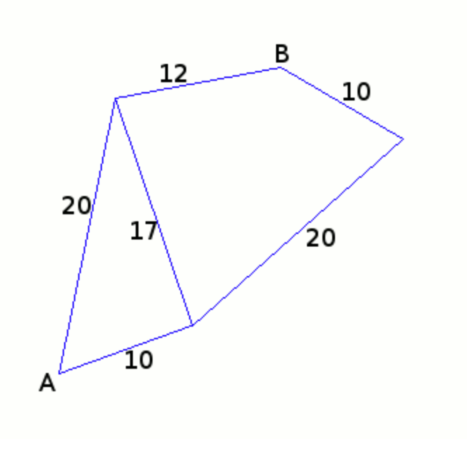
Pathfinding In Games

- Initially the concept was only used in RTS
- Now *the* most important AI technique
  - (probably)

- Still can be buggy. See
  - http://www.ai-blog.net/archives/000152.html
  - http://www.youtube.com/watch?v=lw9G-8gL5o0&feature=player_embedded

Tackling Paths

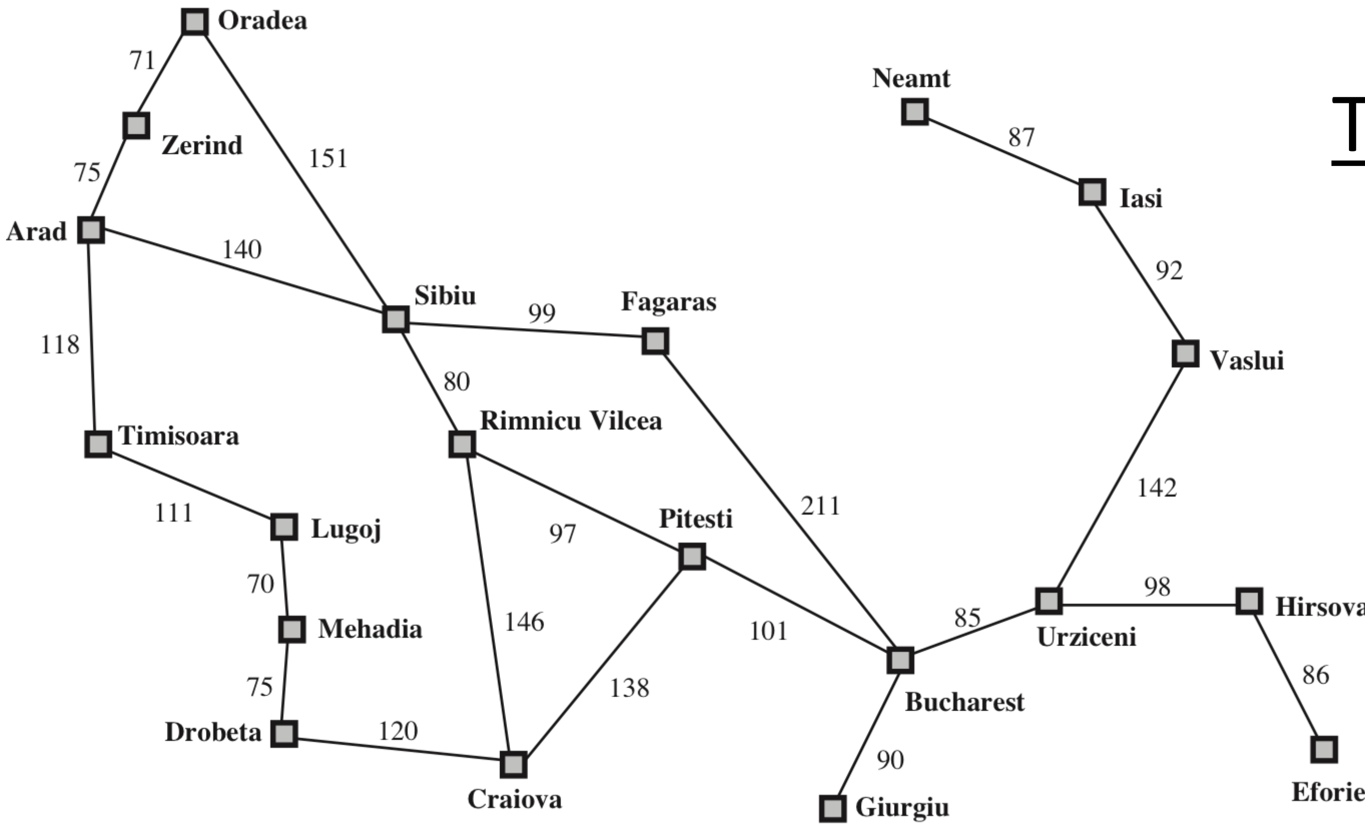
- Characters "live" in a computer world
  - Even developers may not know exact location
    - Physics simulations



- Pathfinders operate on discrete structures

Remember This?

Romania Map



Task:
navigate
from A to B

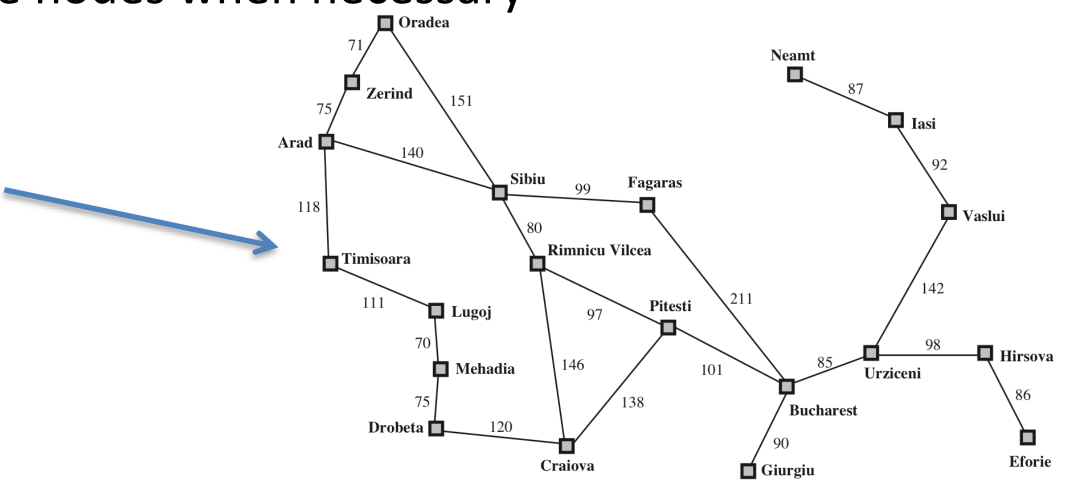
From COMP219:

- A *search* algorithm can solve the navigation problem
- Simple algorithms
  - Breadth-first, depth-first, unit cost,…
    do not work in real-world problems

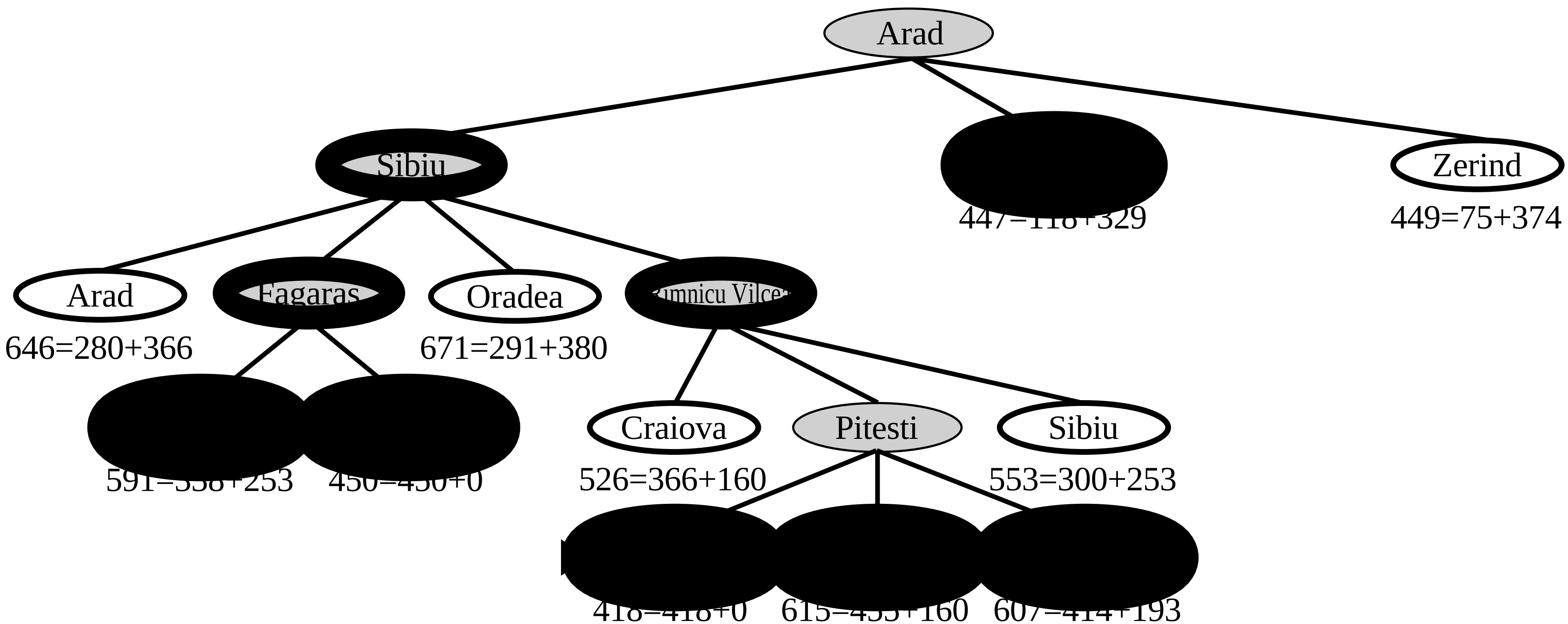
- A* is the best we have

So

- A* works on ***weighted graphs***
  - Pathfinding graphs
  - Explicitly or implicitly represented
    - Romania map: explicit representation
    - Many games **do not** store full graphs
      - Generate nodes when necessary
  - GraphNode



Recall: Search Tree



- An imaginary tree showing all possible states reachable from the initial state
- Search ***strategy*** defines an expansion order

Recall: A* Search (Strategy)

- Combine uniform cost search and greedy search.
- Uses ***heuristic*** f:

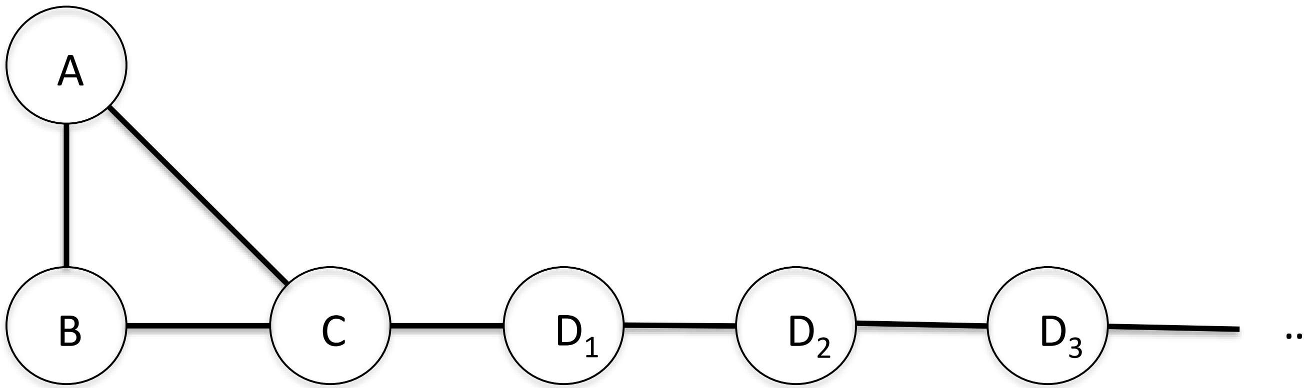
$$f(n) = g(n) + h(n),$$

- where
  - g(n) is path cost of n;
  - h(n) is expected cost of cheapest solution from n.

Recall: General Algorithm for A* Search

```
agenda = initial state;
while agenda not empty do
   take node from agenda such that
     f(node) = min { f(n) | n in agenda}
     where f(n) = g(n) + h(n)
   if node is goal state then
  return solution;
   new nodes = apply operations to node;
   add new nodes to the agenda;
```

Theory V Practice:
Visiting nodes twice



The general framework allows to visit nodes more than once

- **Closed** nodes list: already visited nodes

Theory V Practice: Admissible and Inadmissible Heuristics

- A* is guided by heuristic
- If heuristic is too high (overestimates)
  - It's ***inadmissible***
  - A* is not guaranteed to find best path
  - Does not mean you cannot use it!
    - Faster search vs better paths balance
  - Closed nodes can be "reopened"

A* Requires

- To store the agenda
  - *Open nodes* list
- To store the
  - *Closed* nodes list

- For every open node: costs so far and estimated costs
- For every closed node the *connection* (edge) leading to it

Pathfinding Algorithm

```
while lowest rank in open is not goal
  current = remove lowest rank item from open;
  closed.add(current);
  for neighbors of current:
   Ncost = g(current) + cost(current, neighbor);
   if (open.contains(neighbor)&&Ncost<g(neighbor))
      open.remove(neighbor)
   if (closed.contains(neighbor)&&Ncost<g(neighbor))
      closed.remove(neighbor)
   if (!open.contains(neighbor)&&
                    !closed.contains(neighbor))
      g(neighbor) = Ncost
      open.add(neighbour)
      neighbor.connection = current
```

Good Practice: Class GraphNode

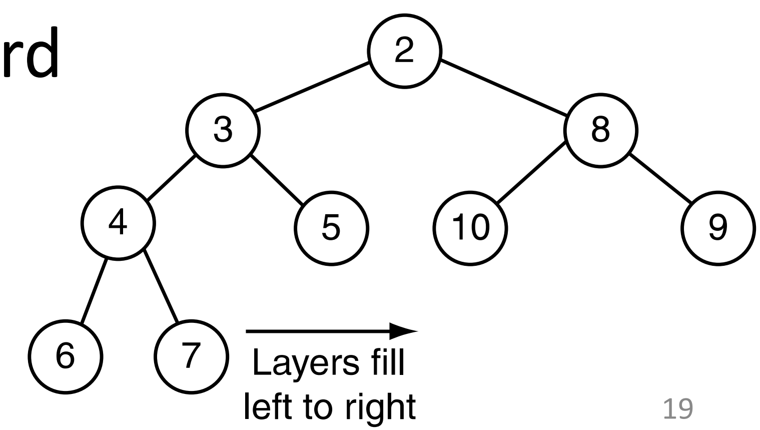
```
public class GraphNode {
  // link to game world
  Vector<Edge> edges
}
public class Edge {
  GraphNode from, to;
  float cost;
}
```

Good Practice: NodeRecord

```
public class NodeRecord {
  GraphNode node;
  Edge connection;
  float costSoFar;
  float estimatedGoalCost;
  float currentCost;
}
```

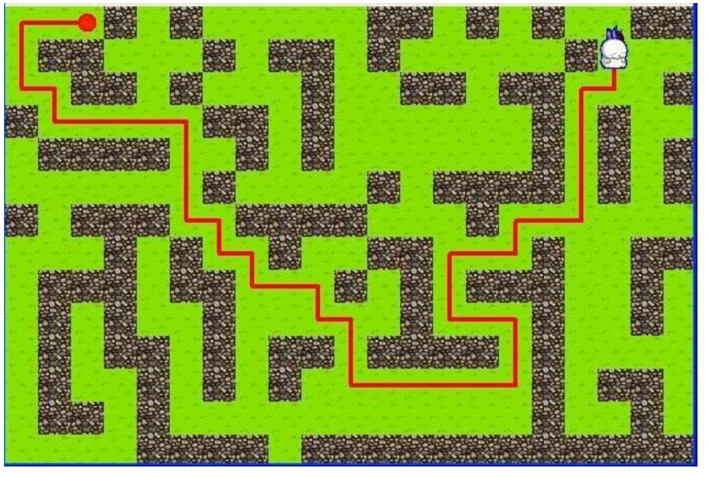
Data Structures

- **Closed**: unsorted list of NodeRecord
- **Open**
  - Unsorted list of NodeRecord
    - Insert: easy (just append)
    - Take: hard (loop through all of them)
  - Priority queue of NodeRecord
    - Insert: medium (balancing)
    - Take: medium



Simplicity Rules

- On a grid-like graph
  - One take per 8 inserts
- With a good heuristics
  - A simple unsorted list might be more efficient than a sophisticated Priority Queue!

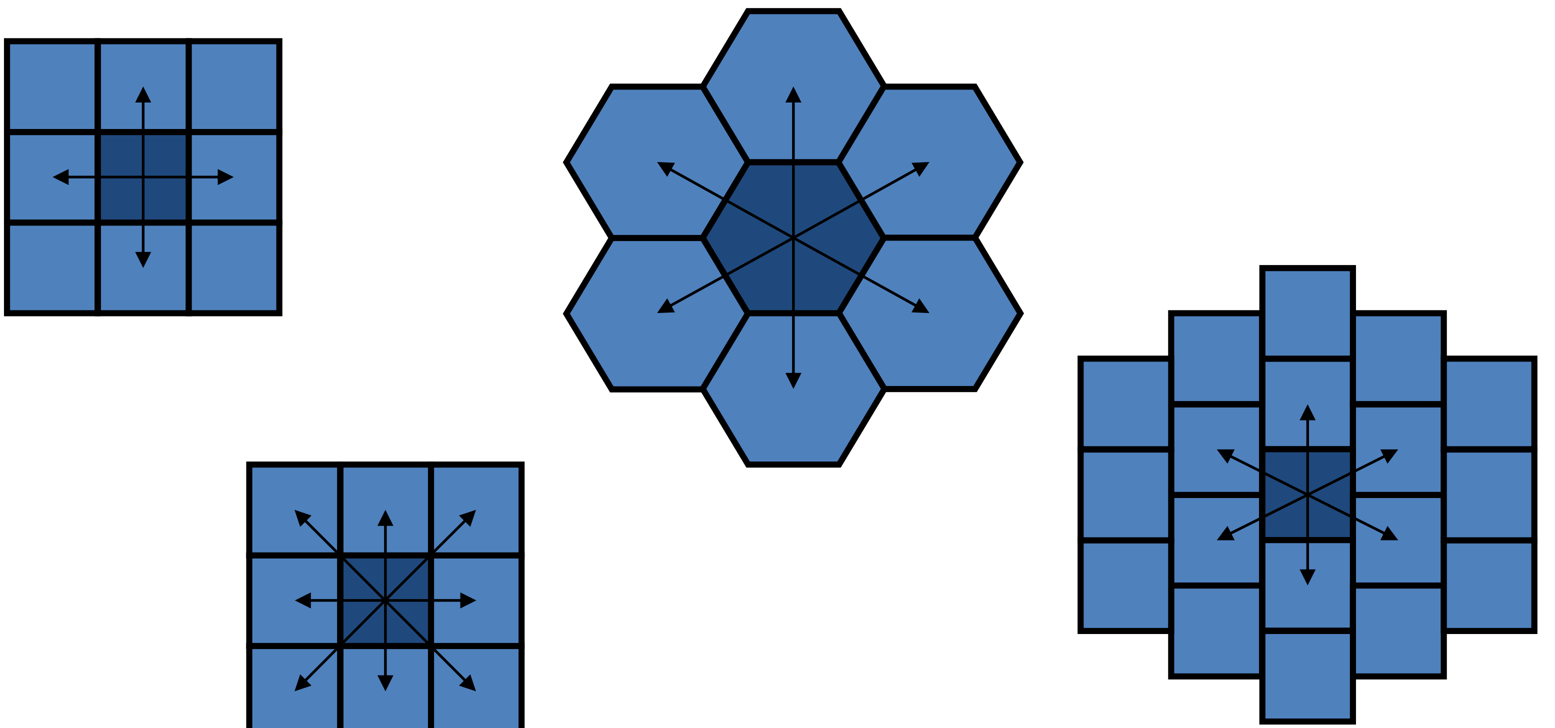


Tile-Based Games

- A vast majority of RTS games are tile-based
  - Every unit occupies (one or more) tile
  - Every tile can accommodate ≤ 1 unit
- A tile is either **blocked** or **passable**

Tile Shapes

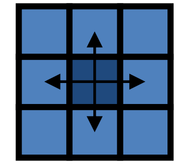
- Different games use different tiles



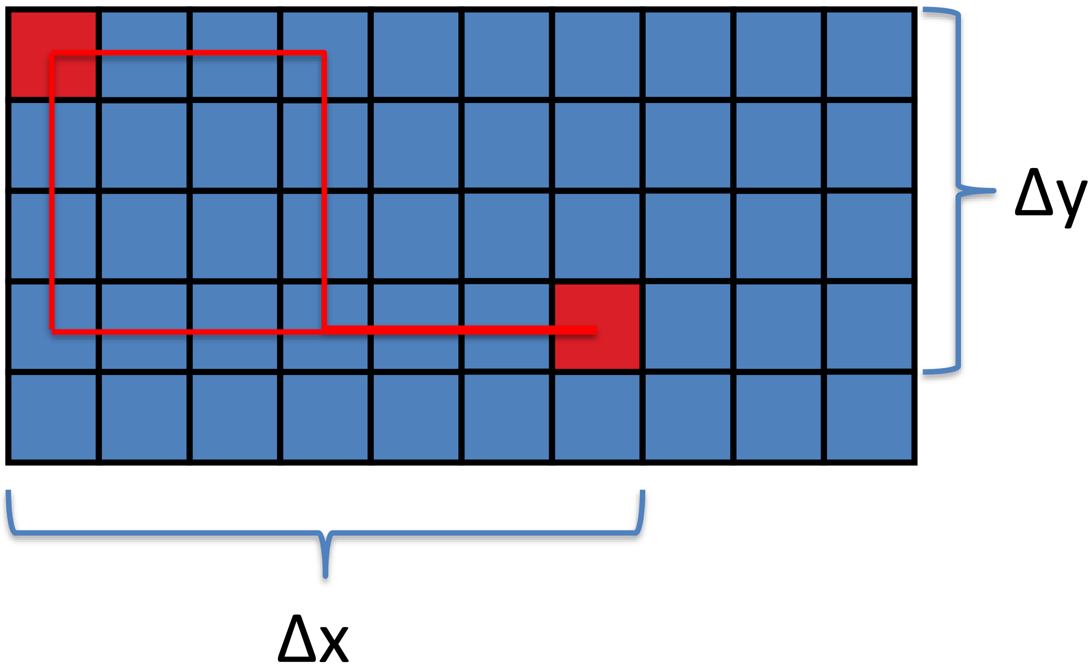
Nodes

- A node is uniquely identified with (x,y) coordinates
- No need to store neighbour nodes
  - Easily compute when necessary

Heuristics



- Manhattan block distance: Δx + Δy



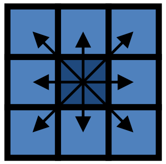
Trouble: too many paths of same value

Breaking Ties

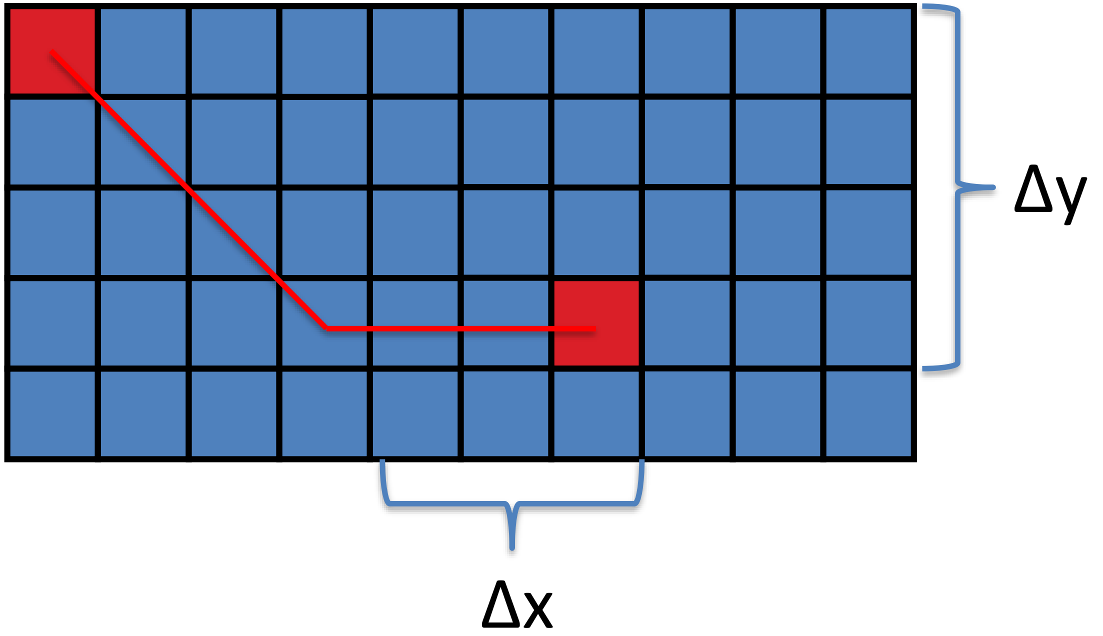
- Breaking ties is one of the reasons to consider an *inadmissible heuristics:*
  - *Biased towards pursuing the goal*
  - A* can run faster
  - If it is just slightly higher, A* will still find best paths

- Other reason?
  - Distance in hours, heuristics in km
  - Computational complexity

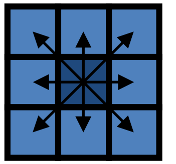
Heuristics



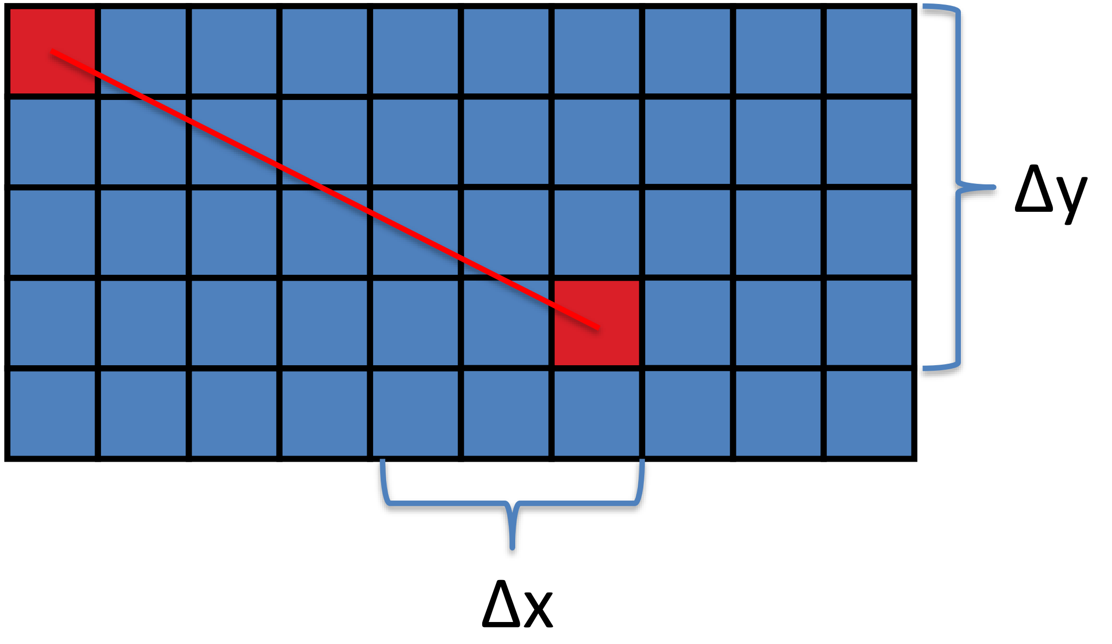
- Diagonal moves allowed: **Δx + Δy**



Heuristics

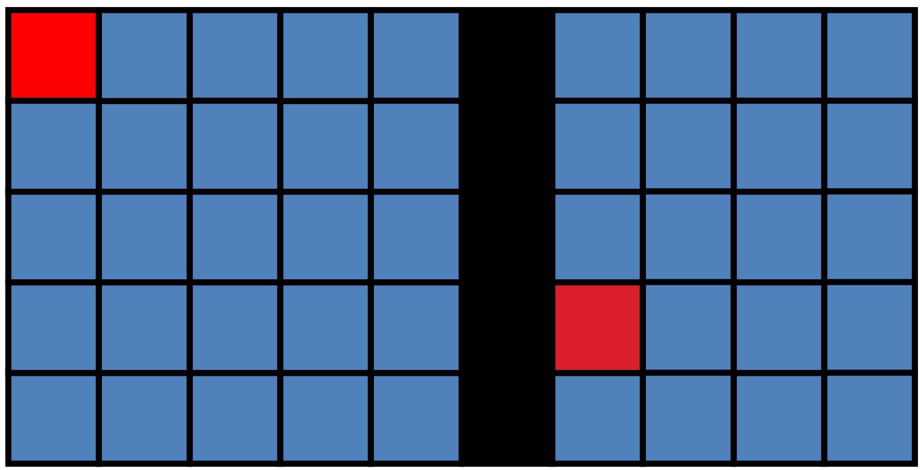


- Euclidian distance: $\sqrt{(Dx)^2 + (Dy)^2}$



Worst Possible Case

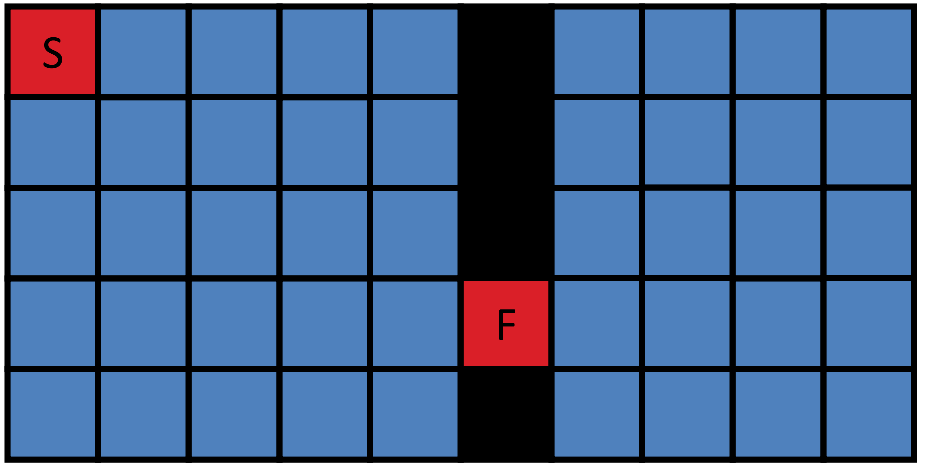
- Worst possible case for any search algorithm
  - No path



  - Will explore all available space

Updated Pathfinding

- Check if Start and Finish are valid locations
  - If Finish is not valid, no path
  - If **Start** is not valid
    - Something goes wrong
    - Delete agent?
    - Move to a valid location?
    - …



Zone Mapping

- Every tile belongs to a *zone*
  - 0 – impassable
  - Same number – can pass

1	1	1	1	1	0	2	2	2	2
1	1	1	1	1	0	2	2	2	2
1	1	1	1	1	0	2	2	2	2
1	1	1	1	1	0	2	2	2	2
1	1	1	1	1	0	2	2	2	2

  - Zone equivalence array
    - Hovercraft[0]=0; Hovercraft[1]=0; Hovercraft[2]=0

Zone Equivalence Array

- For every zone number and
- Every vehicle class
  - ZEA[zone number]
    - Either zone itself
    - Or the smallest equivalent zone number
- If (ZEA[S.zone] == ZEA[F.zone])
  - Call the pathfinder

Pathfinding Pool

- Running an A* algorithm takes time
- In RTS games there are dozens of characters
- If every one of them starts A*…
  - A *pool* of pathfinders
  - A queue of agents waiting for paths

  - Start moving / play animation while waiting