Game AI: The set of algorithms, representations, tools, and tricks that support the creation and management of real-time digital experiences

_____: A rule of thumb, simplification, or educated guess that reduces or limits the search for solutions in domains that are difficult and poorly understood.

Grid Generation Hints

- If no world line goes through grid lines (rayTraceWorld)
- If no obstacle point within grid cell (pointInsidePolygonPoints, for each obst.)
- Please check the following sections
  - Miscellaneous utility functions
  - Hints

PREVIOUSLY ON…

Class N-2

1. How would you describe AI (generally), to not us?
2. Game AI is really about
   - The I____ of I_____. Which is what?
   - Supporting the P_____ E_____ which is all about… making the game more enjoyable
   - Doing all the things that a(nother) player or designer…
3. What are ways Game AI differs from Academic AI?
4. (academic) AI *in* games vs. AI *for* games. What's that?
5. What is the complexity fallacy?
6. The essence of a game is a g_ a set of r_, and a__?
7. What are three big components of game AI in-game?
8. What is a way game AI is used out-of-game?

Class N-1

1. What is attack “kung fu” style?
2. How do intentional mistakes help games?
3. What defines a graph?
4. What defines graph search?
5. Name 3 uniformed graphs search algorithms.
6. What is a heuristic?
7. Admissible heuristics never ___estimate
8. Examples of using graphs for games

Sidebar: Iterative Deepening

- mid 1970s
- Idea: perform depth-limited DFS repeatedly, with an increasing depth limit, until a solution is found.
- Each repetition of depth-limited DFS needlessly duplicates all prior work?
  - Duplication is not significant because a branching factor b > 1 implies that the number of nodes at depth k exactly is much greater than the total number of nodes at all depths k-1 and less.
- That is: most nodes are in the bottom level.

Number of nodes

- Full, complete, balanced binary tree, height *h*
  - Number of nodes *N* = 2^{h+1} – 1
    - Height 0: 2^0 = 1
    - Height 1: 2^1 = 2
    - Height 2: 2^2 = 4
    - Height 3: 2^3 = 8
    - N = 1 + 2 + 2^2 + 2^3 + … + 2^h
      = (2^{h+1} – 1) / (2 – 1) = 2^{h+1} - 1
  - Number of leaves *L* = 2^h
  - Height 42, *N = 8,796,093,022,207*
    L = 4,398,046,511,104

BRIEF RECAP OF LAST TIME

Path finding models

1. Tile-based graph – “grid navigation”
2. Path Networks / Points of Visibility NavGraph
3. Expanded Geometry
4. NavMesh

Model 1: Grid Navigation

- 2D tile representation mapped to floor/level
  - Squares, hex; 8 or 6 neighbors / connectivity
- Mainly RTS games
- One entity/unit per cell
- Each cell can be assigned terrain type
- Bit mask for non-traversable areas
- Navigation: A*, Dijkstra

Path Planner

- Initial state (cell), Goal state (cell)
- Each cell is a state agent can occupy
- Sort successors, try one at a time (backtrack)
- Heuristic: Manhattan or straight-line distance
- Each successor stores who generated it

Grid navigation: pros

- Discrete space is simple
- Can be generated algorithmically at runtime
- Good for large number of units
- A* works really well on grids (uniform action cost, not many tricky spots)

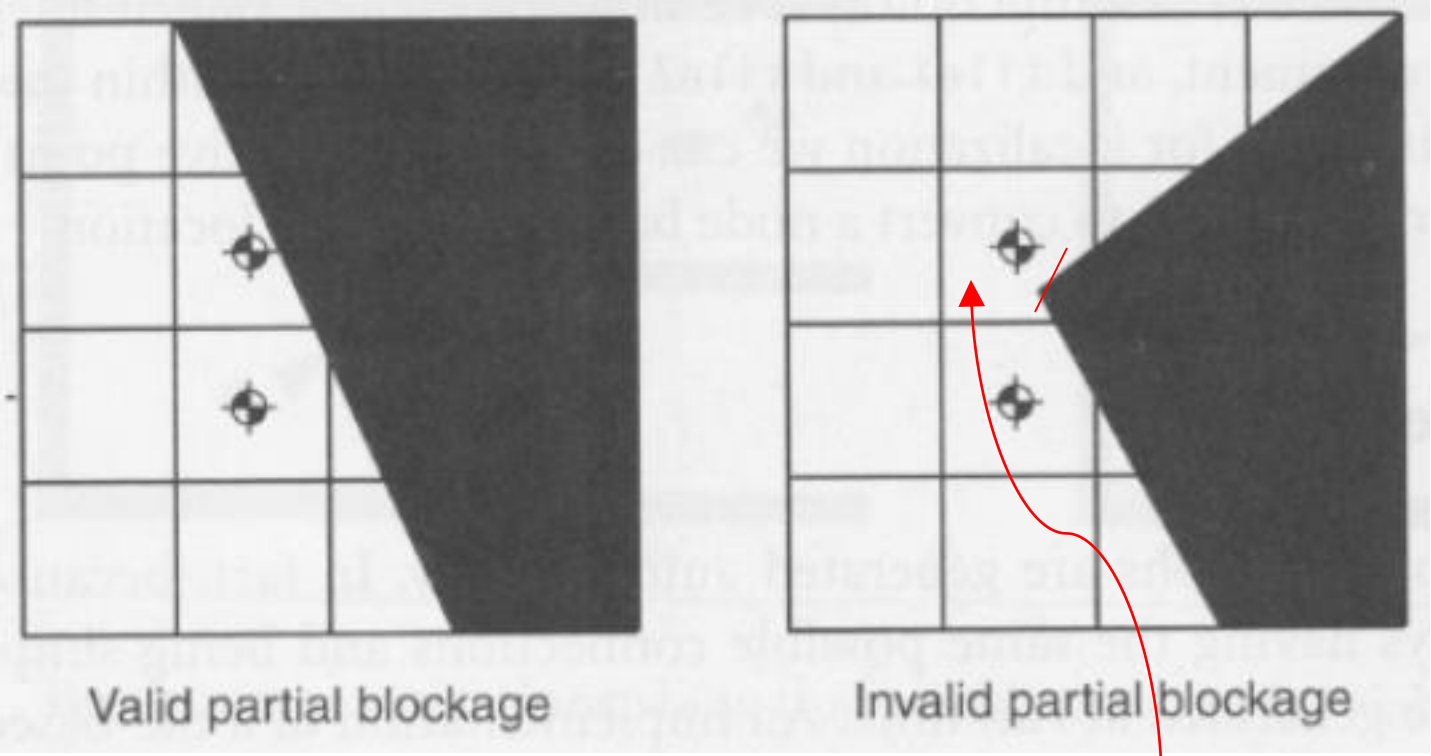
Grid navigation: cons

- Discretization “wastes” space
- Agent movement is jagged/awkward/blocky, though can be smoothed
- Some genres need continuous spaces
- Partial-blocking hurts validity
- Search must visit a lot of nodes (cells)
- Search spaces can quickly become huge
  - E.g. 100x10 map == 100k nodes and ~78k edges

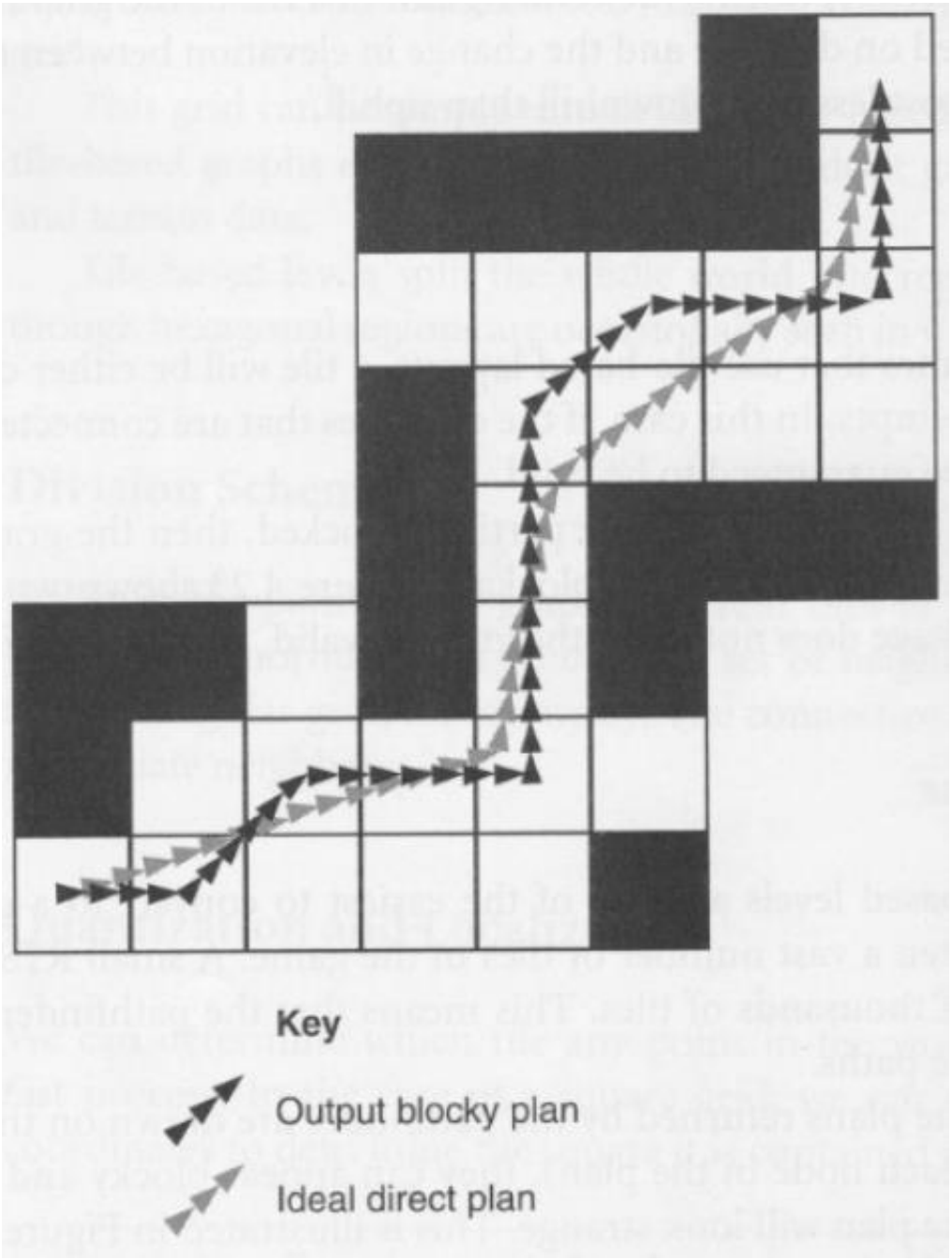
New Problems

- Generation
- Validity
- Quantization
  - Converting an in-game position (for yourself or an object) into a graph node
- Localization
  - Convert nodes back into game world locations (for interaction and movement)

Validity



not all points in a grid square are reachable from each other!

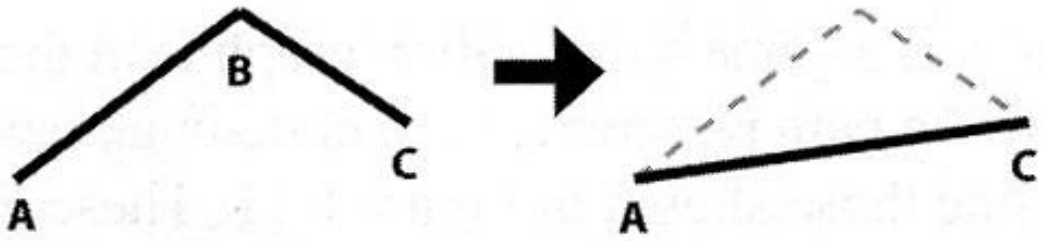


- String pulling
- Splines
- Hierarchical A*

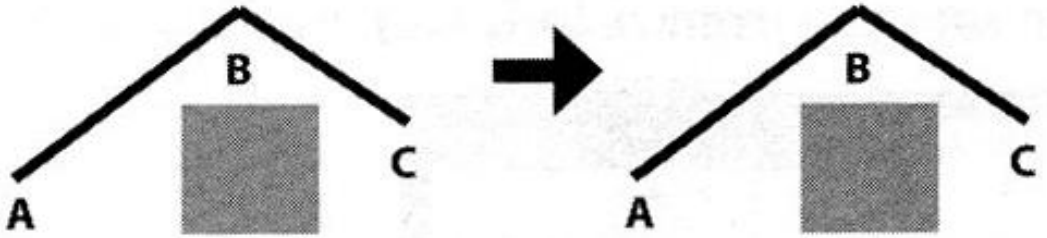
Path Smoothing

- Zig-zagging from point to point looks unnatural

- Post-search smoothing can elicit better paths



There is no obstacle obstructing the path from A to C so the two edges can be replaced with one.



With an obstacle in the way both edges are necessary

Quick Path-Smoothing

- Given a path, look at first two edges, E1 & E2
  1. Get E1_src and E2_dest
  2. If unobstructed path between the two, set E1_dest = E2_dest, then delete E2 from the path. Set next edge as E2.
  3. Else, increment E1 and E2.
  4. Repeat until E2_dest == goal.

Slow Path-Smoothing

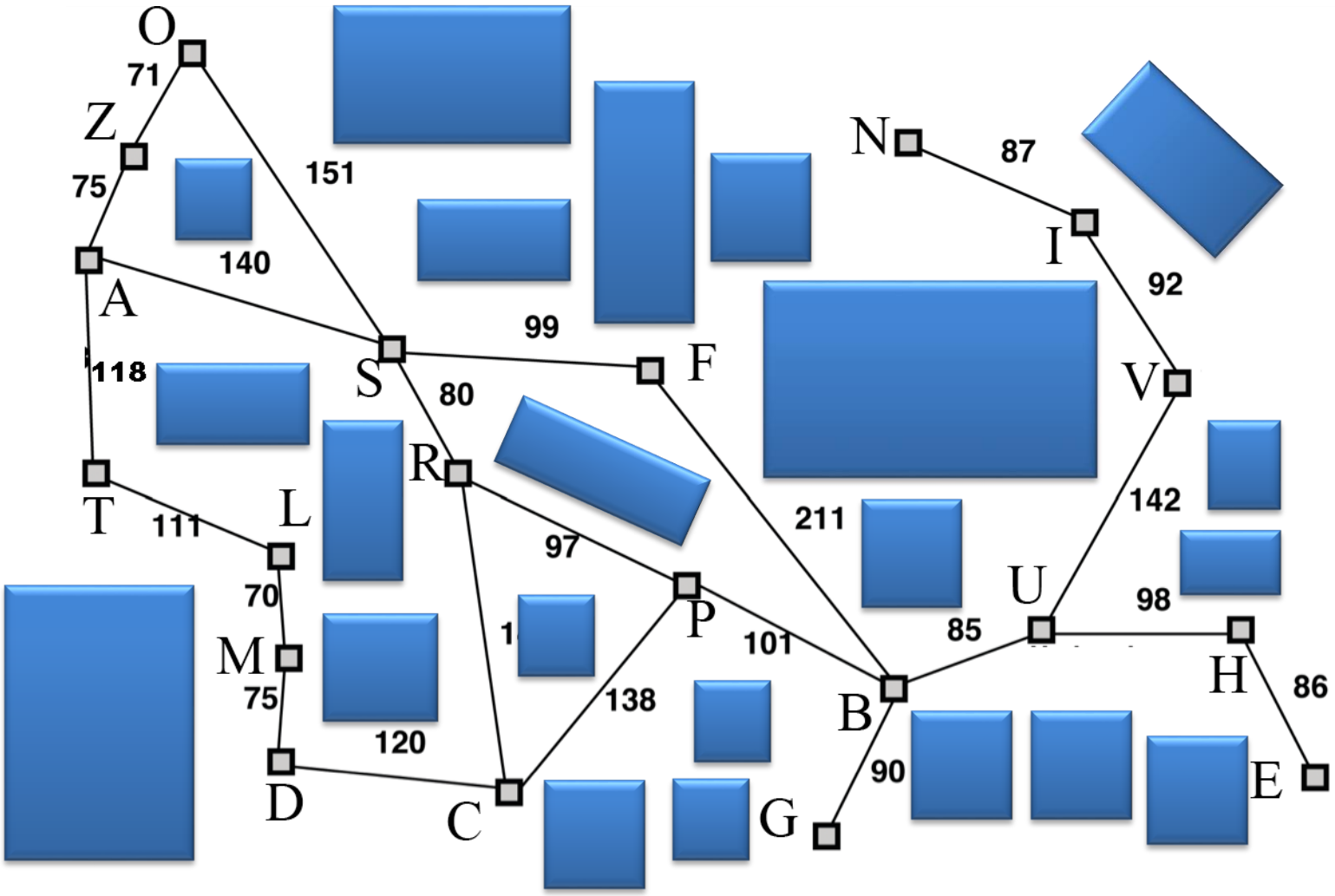
- Given a path, look at first two edges, E1 & E2
  1. Get E1_src and E2_dest
  2. If unobstructed path between the two, set E1_dest = E2_dest, then delete E2 from the path. Set E1 and E2 from beginning of path.
  3. Else, increment E1 and E2.
  4. Repeat until E2_dest == goal.

Graphs, Search, & Path Planning Continued

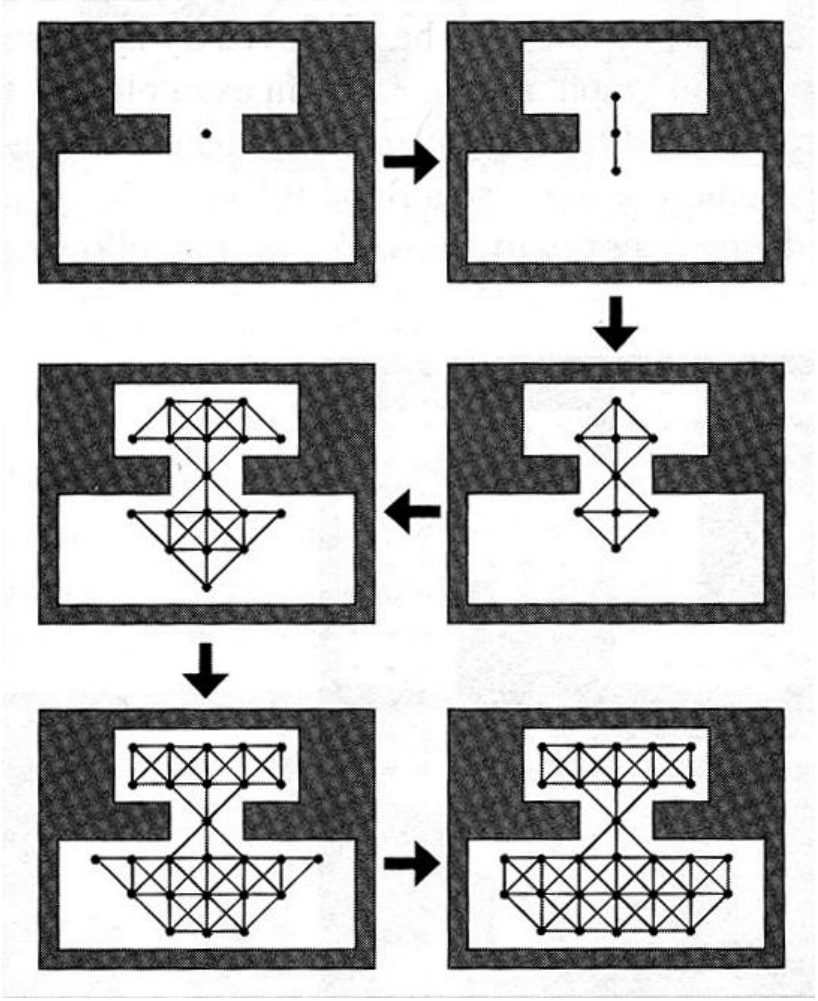
2016-05-24

M2: Path Networks

- POV: Points of visibility NavGraph (see B CH 8)
- Discretization of space into sparse network of nodes
- Two-tiered navigation system
  - Local, continuous
  - Remote
- Connects points *visible to each other* in all important areas of map
- Usually hand-tailored (can use flood-fill)


O
71
Z
75
A
140
151
99
S
118
80
F
N
87
I
92
V
T
111
L
R
97
211
142
70
M
75
120
D
C
P
101
138
U
85
98
H
86
B
90
G
E

Flood Fill



- Start with “seed”
- “grow” graph
- Designer can move, delete, or add nodes
- Ensure all nodes and edges are at least as far from walls as agents bounding radius

Buckland Figure 8.7

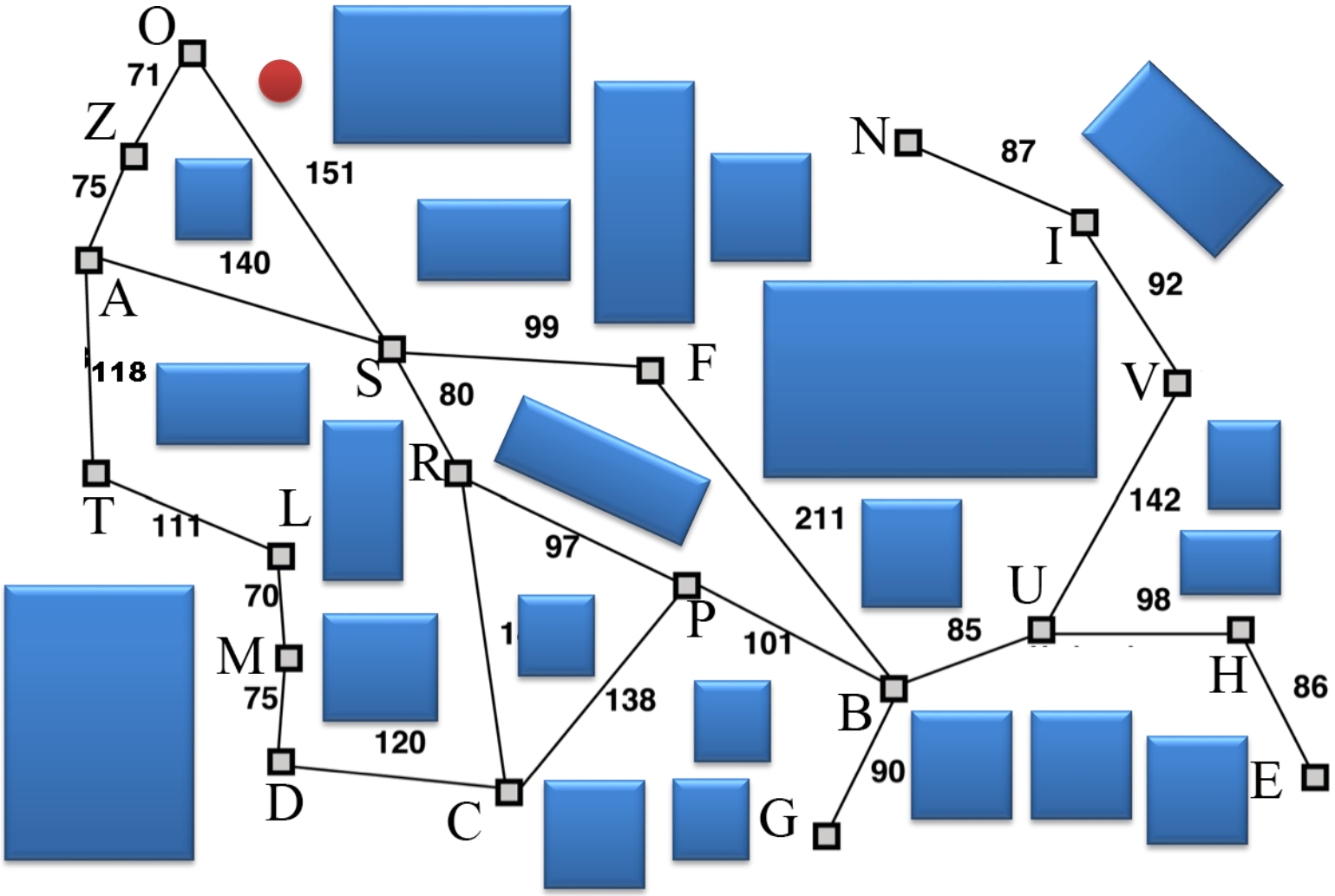
Path network navigation

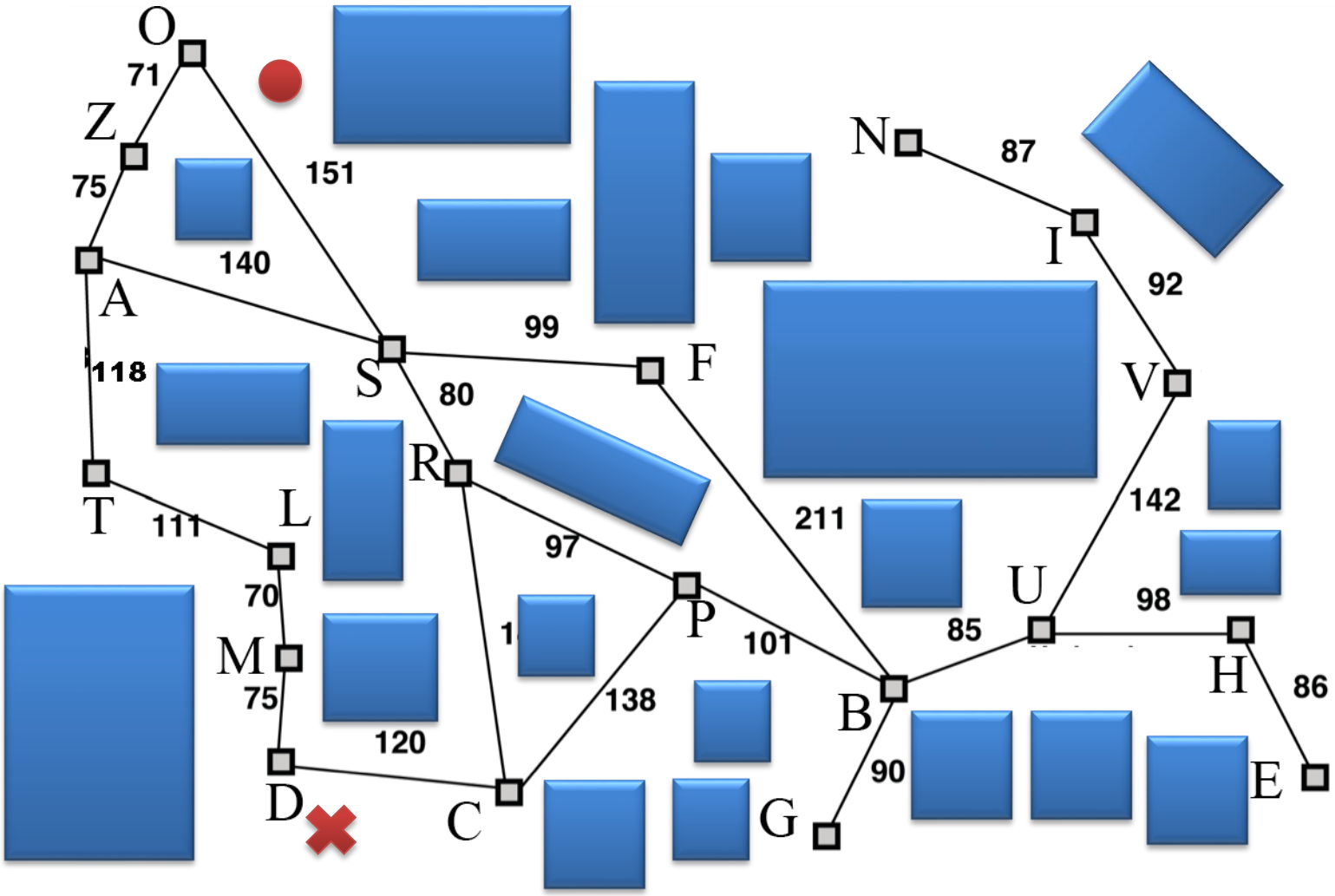
- Path nodes
  - Option 1: manual path node placement
  - Option 2: automatic path node placement
- What happens if you want to go to a place you cannot see?
  - Can’t get there from here
  - Local search
  - Flood fill (increase granularity of nav graph)

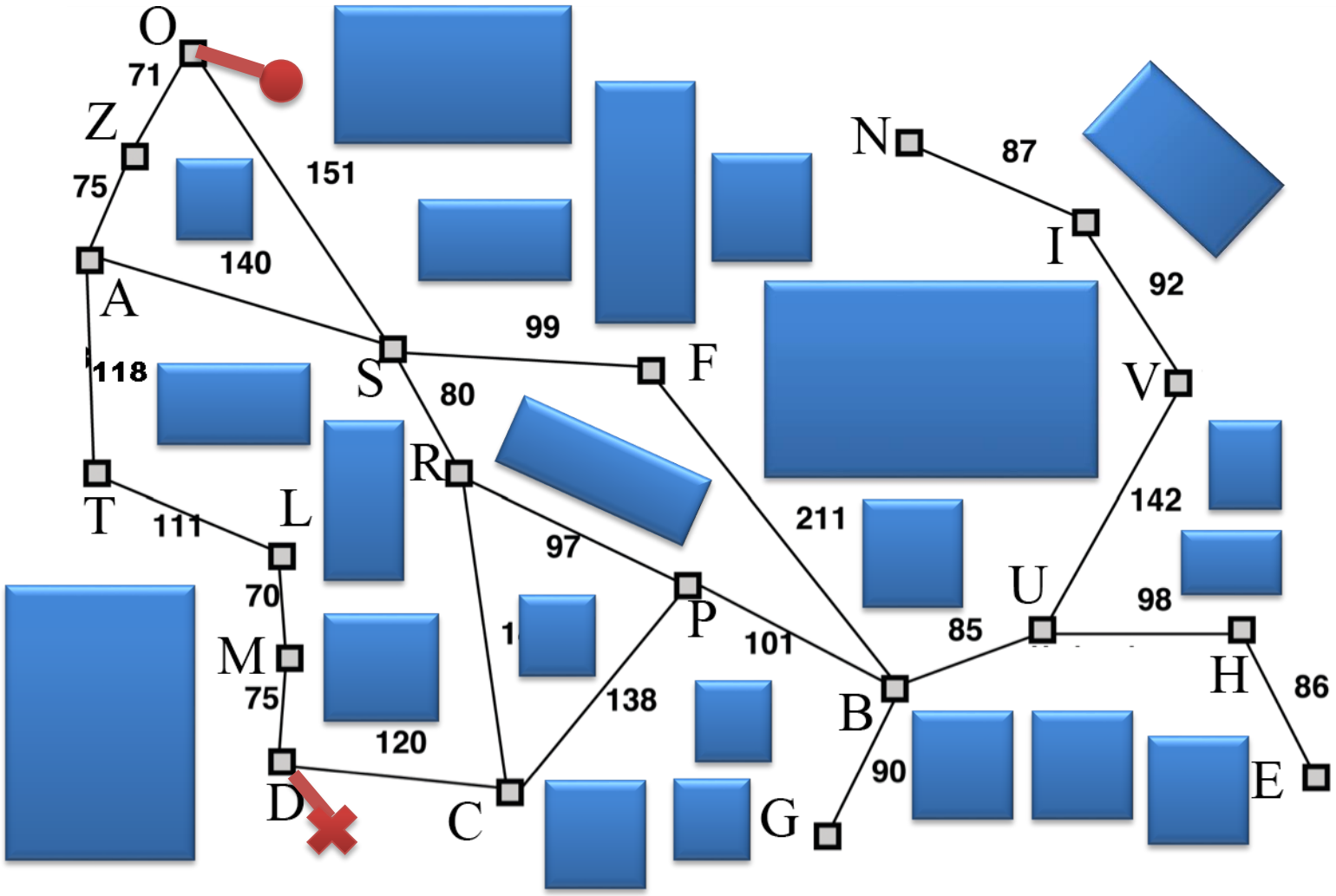
Using the path network

- Basic AI steps when told to go to target X
  1. Find the closest visible graph node (A)
  2. Find the closest visible graph node to X (B)
  3. Search for lowest cost path from A to B
  4. Move to A
  5. Traverse path
  6. Move from B to X

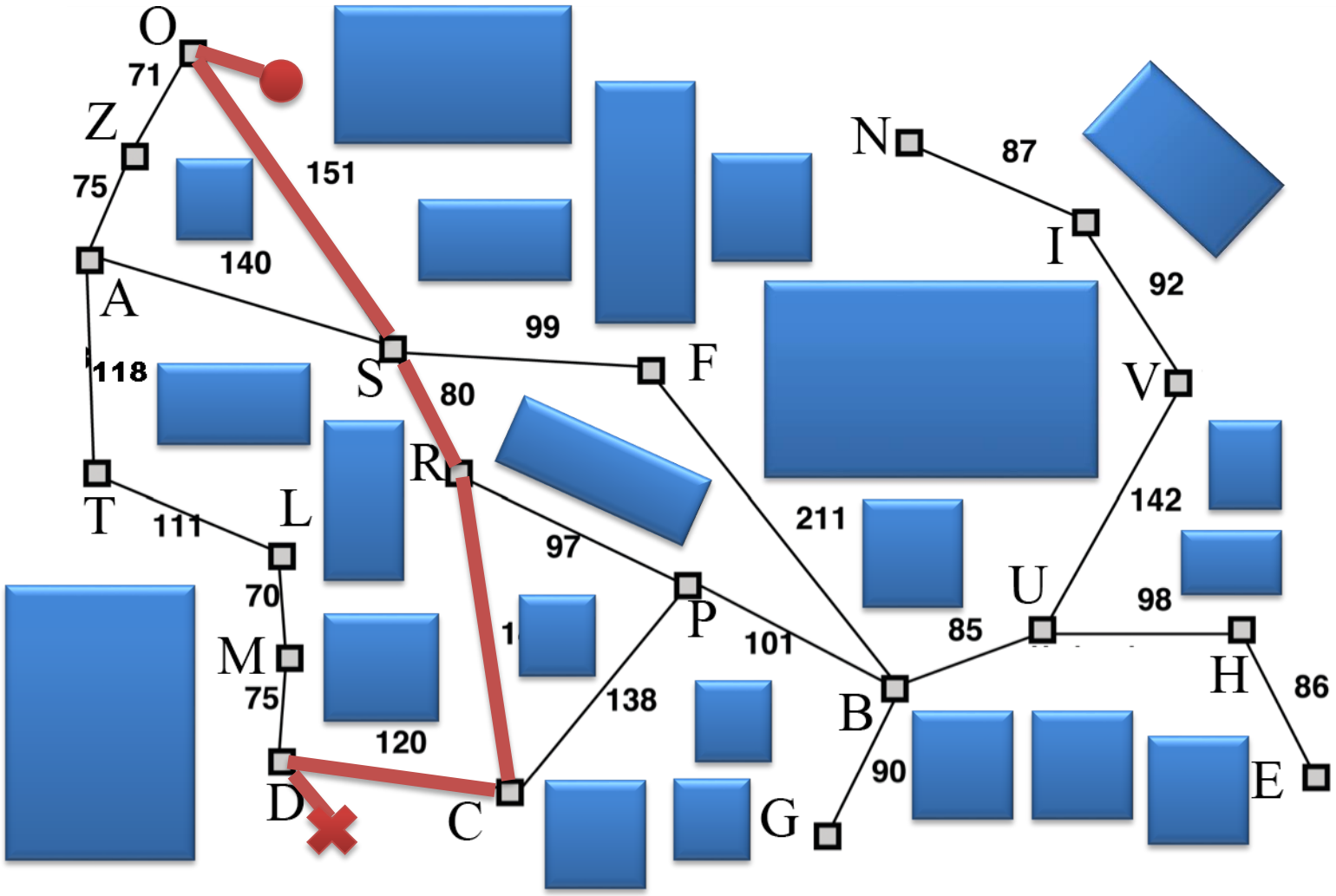
Problem: Unsightly paths.


O
71
Z
75
A
140
151
99
S
118
80
F
N
87
I
92
V
T
111
L
R
97
211
142
70
M
75
120
D
C
P
101
138
U
85
98
H
86
B
90
G
E

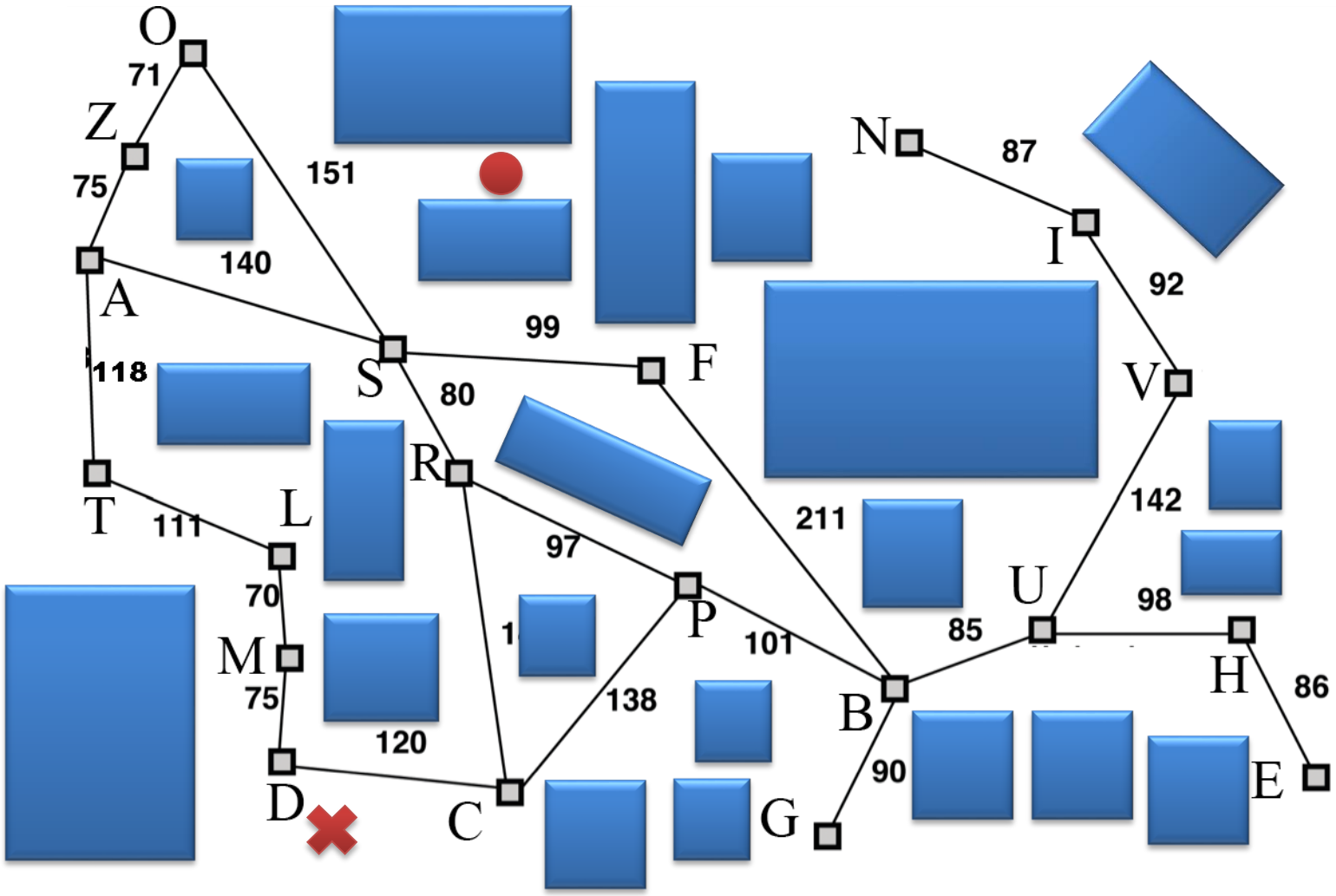

O
71
Z
75
A
140
151
99
S
118
80
F
N
87
I
92
V
T
111
L
R
97
211
142
70
M
75
120
1
P
101
85
U
98
H
86
B
138
90
E
D
C
G

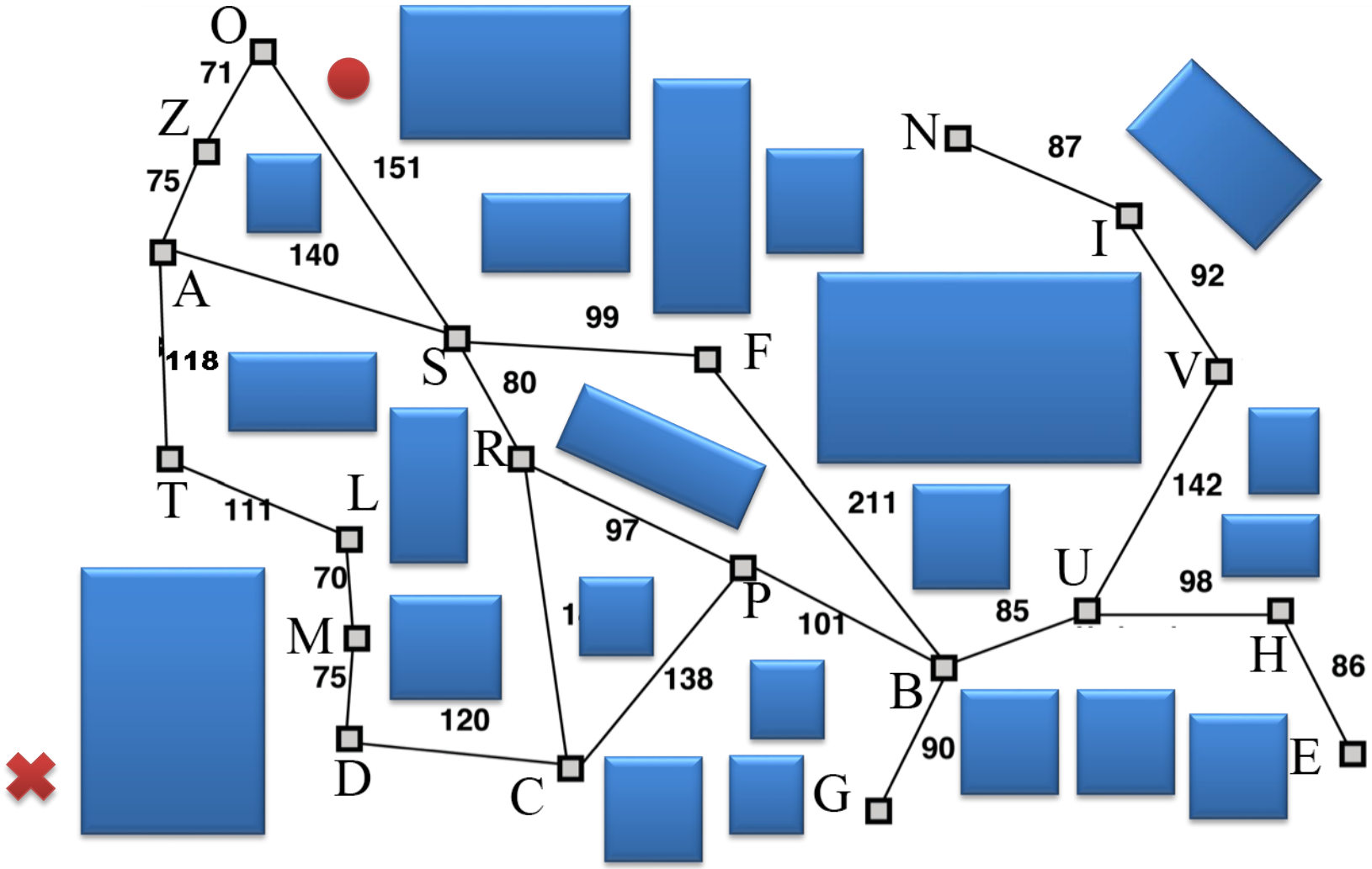

O
71
Z
75
A
140
151
99
S
80
F
118
T
111
L
70
R
97
P
101
211
N
87
I
92
V
142
U
85
98
H
86
E
M
75
120
138
B
90
D
C
G
1

Unnecessary double-backs

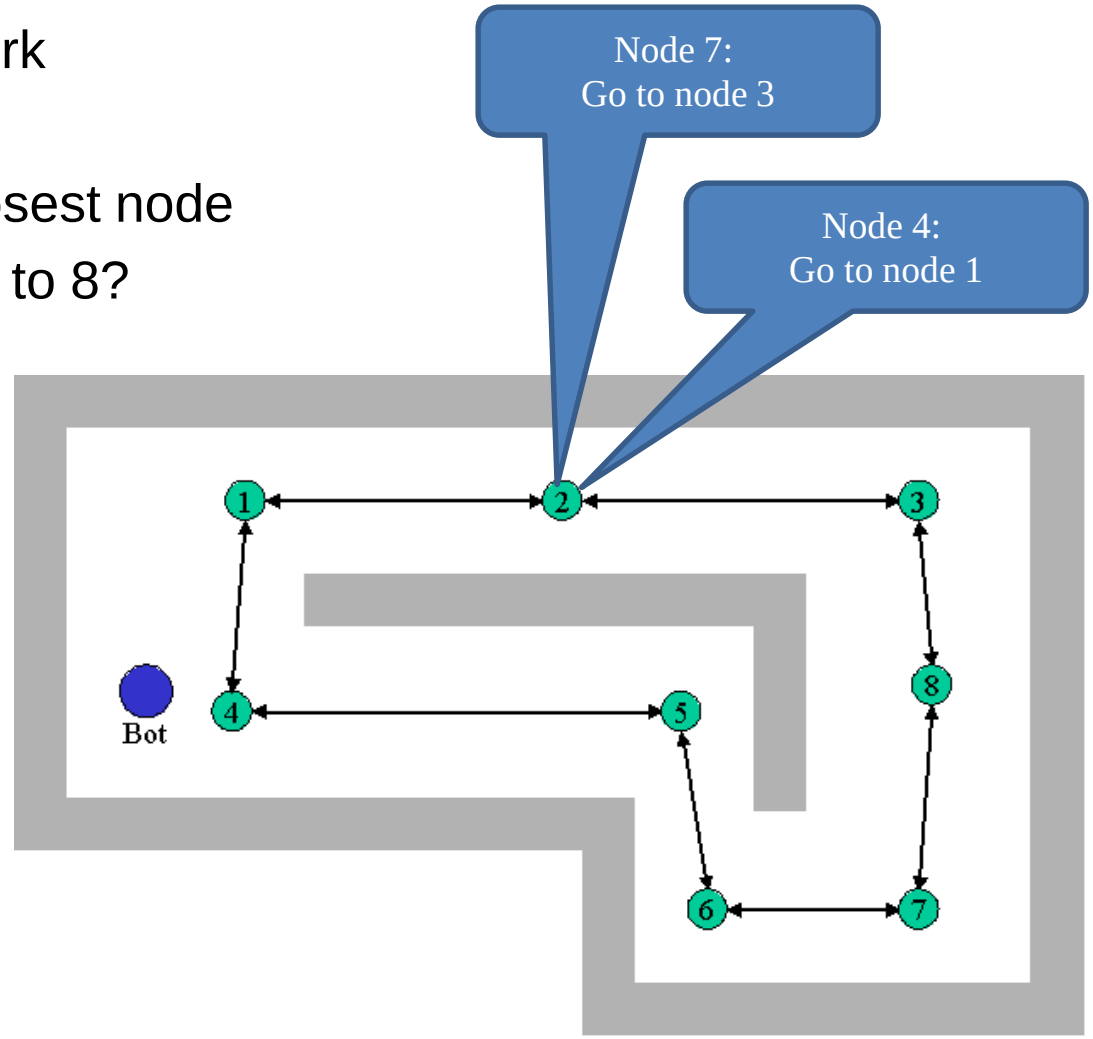


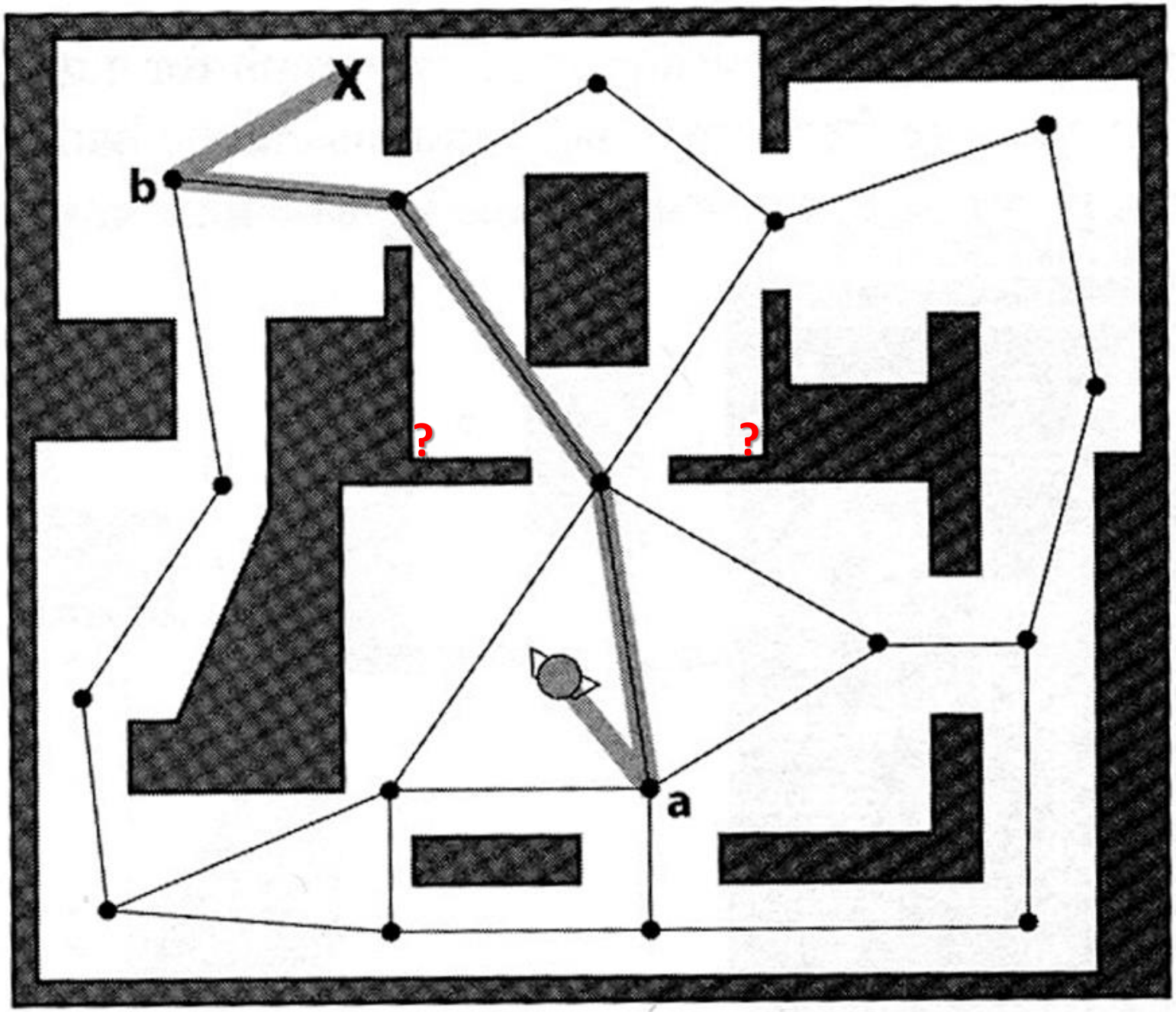
Going where you can't see

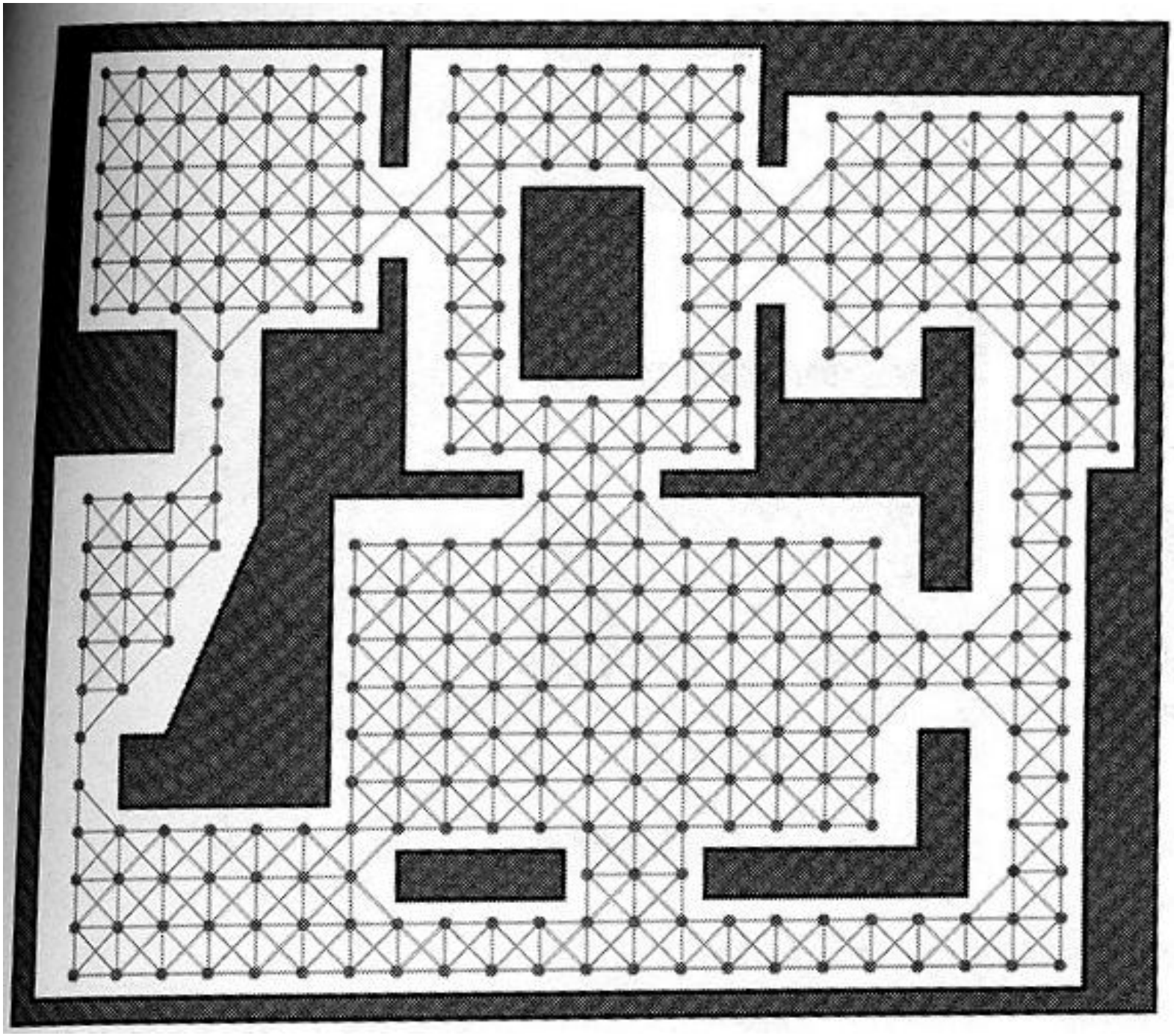



O
71
Z
75
A
140
151
99
S
118
80
F
N
87
I
92
V
T
111
L
R
97
211
142
70
M
75
120
D
C
P
101
138
U
85
98
H
86
B
90
G
E

- Using a path node network
- Bot decides to go to 8
- Get on the network at closest node
- Ask: where to next to get to 8?
- Global table, or store in nodes themselves




X
b
?
?
a



Create navigation table

- For any two nodes (a, b) tells the agent what node, c to go to next.
- For source node v:
  - For each node u:
    - Follow the parent links until you get to v
    - Record the last node before getting to v

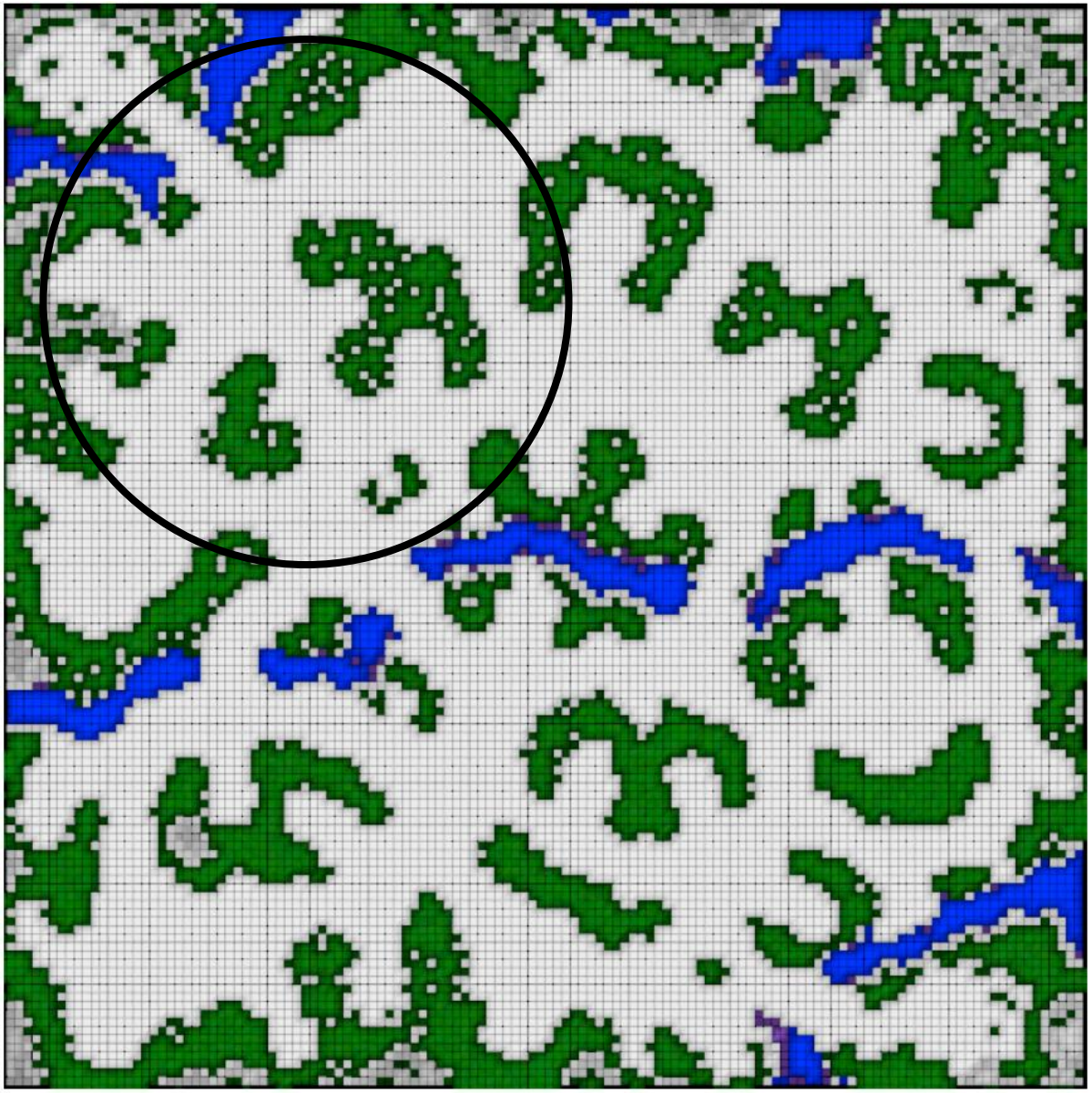
- Dijkstra running time: $O(|V|^2)$ * Can run in $O(|E| + |V|\log|V|)$
- Fully path node process: $O(|V|^3)$ * Run Djikstra |V| times.

Path network: pros

- Discretization of space is very small
- Does not require agent to be at one of path nodes at all times (unlike grid)
- Can be used with navigation mesh to automatically identify where to place path nodes
- Continuous, non-grid movement in local area
- Switch between local and remote navigation
- Plays nice with “steering” behaviors
- Good for FPS, RPGs
- Can indicate special spots (e.g. sniping, crouching, etc.)

Path network: cons

- https://www.youtube.com/watch?v=WzYEZVI46Uw
- Getting on and off the network can be awkward
- Path node placement
  - Difficult for complex maps
  - May have invisible spots
- Dynamic pathing in destructable terrain
- Doesn’t fit well with map-generation features in games
- Fog-of-war in RTSs



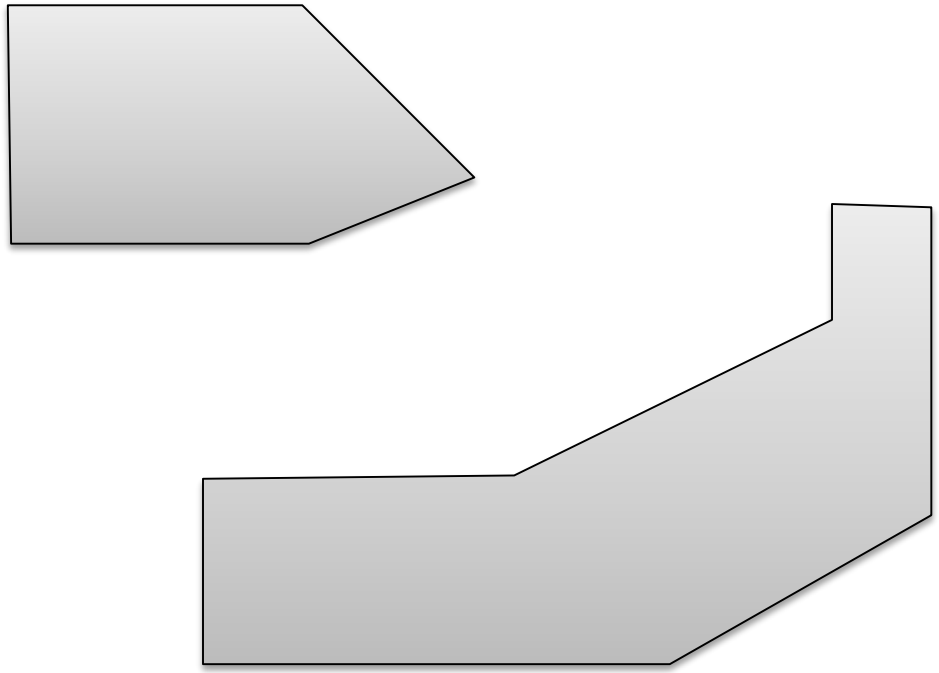
Fog of war

M3: Expanded Geometry

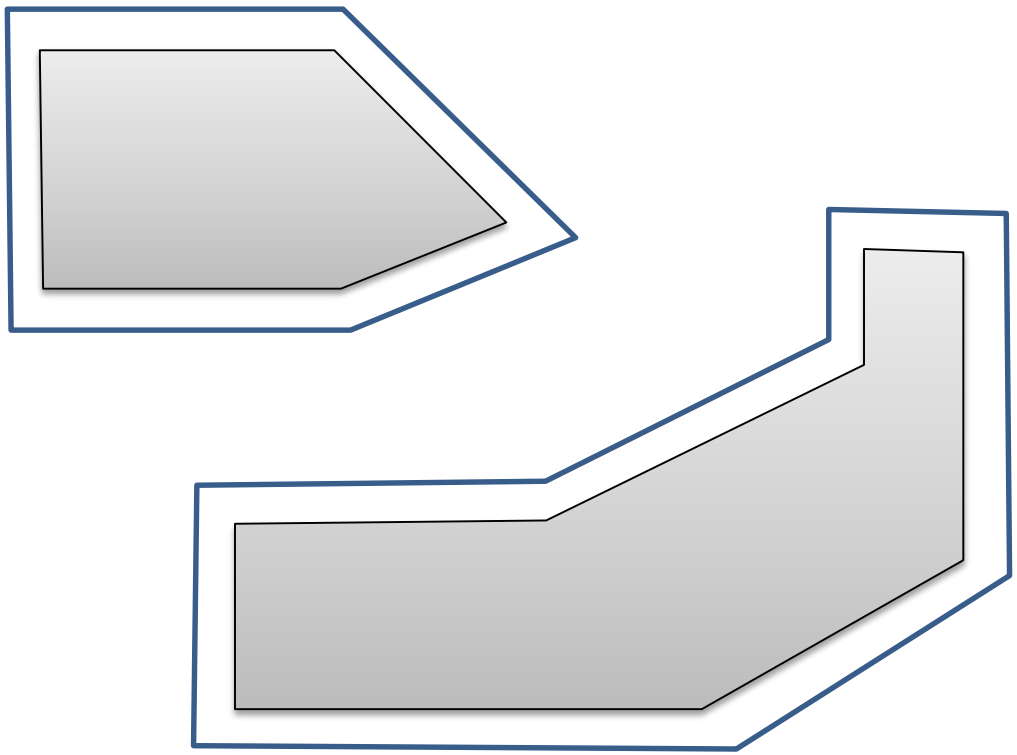
- Automatic, and no wall bumping.

- Automatically expand boundaries of obstacles ($\Delta \geq$ agent_radius)
- Add vertices as nodes
- Test line of sight for all vertices ($O(n^2)$)
- Add edges where (v_1, v_2) == true

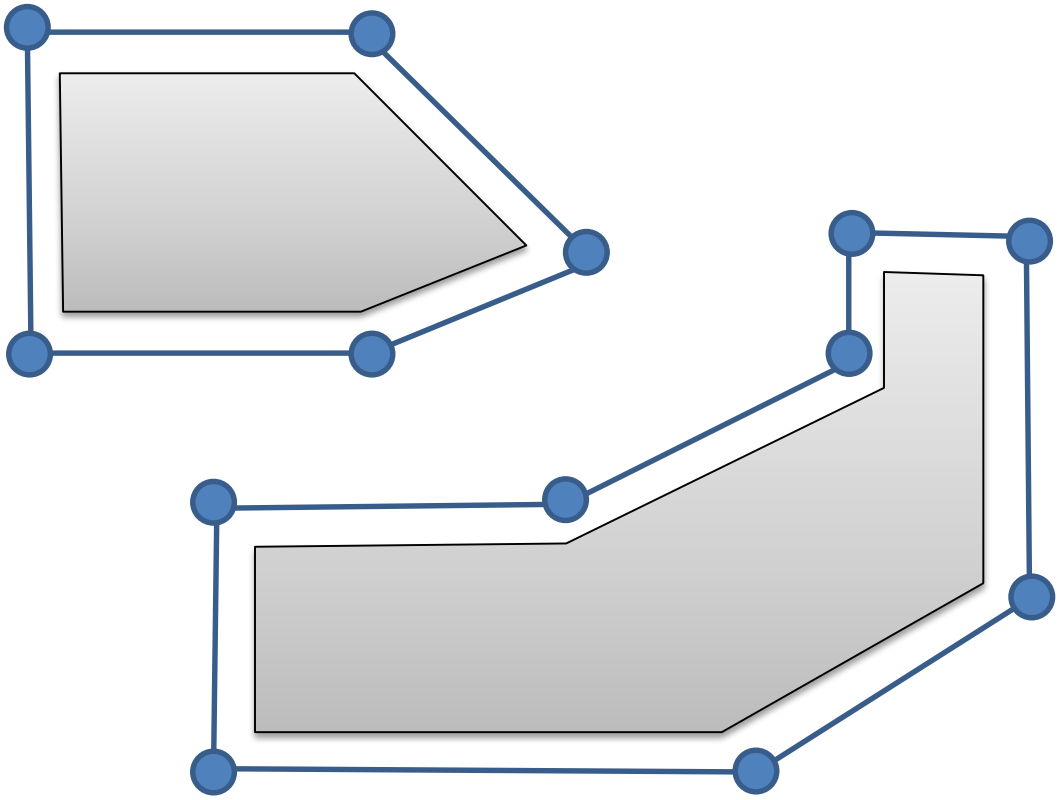
Simple Geometry



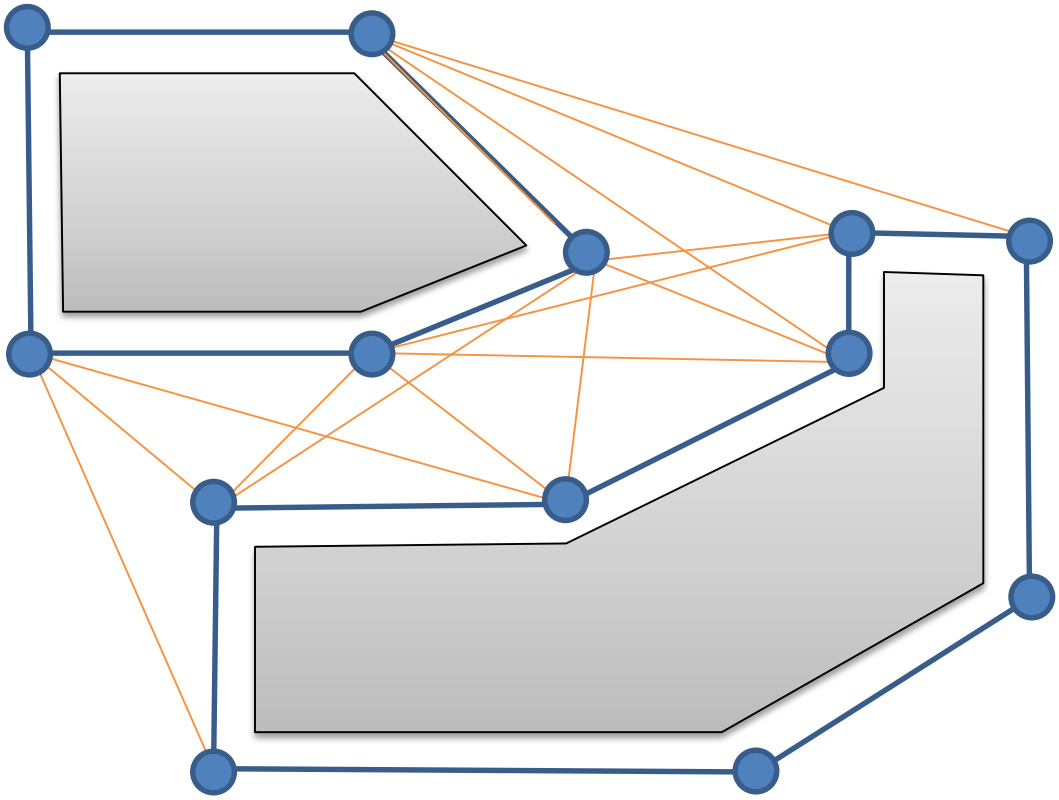
Expanded Geometry



Expanded Geometry



Finished POV Nav Graph



Expanded Geo: pros

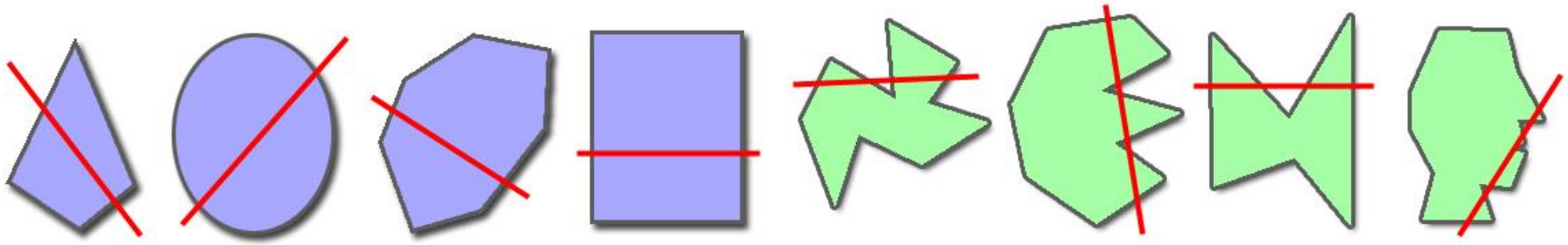
- Discretization of space can be smaller
- Continuous, non-grid movement in local area
- Can work with auto map generation
- Switch between local and remote navigation
- Can play nice with “steering” behaviors

Expanded Geo: cons

- Getting on and off the network can be even more awkward
- Dynamic pathing in destructable terrain
- Fog-of-war in RTSs

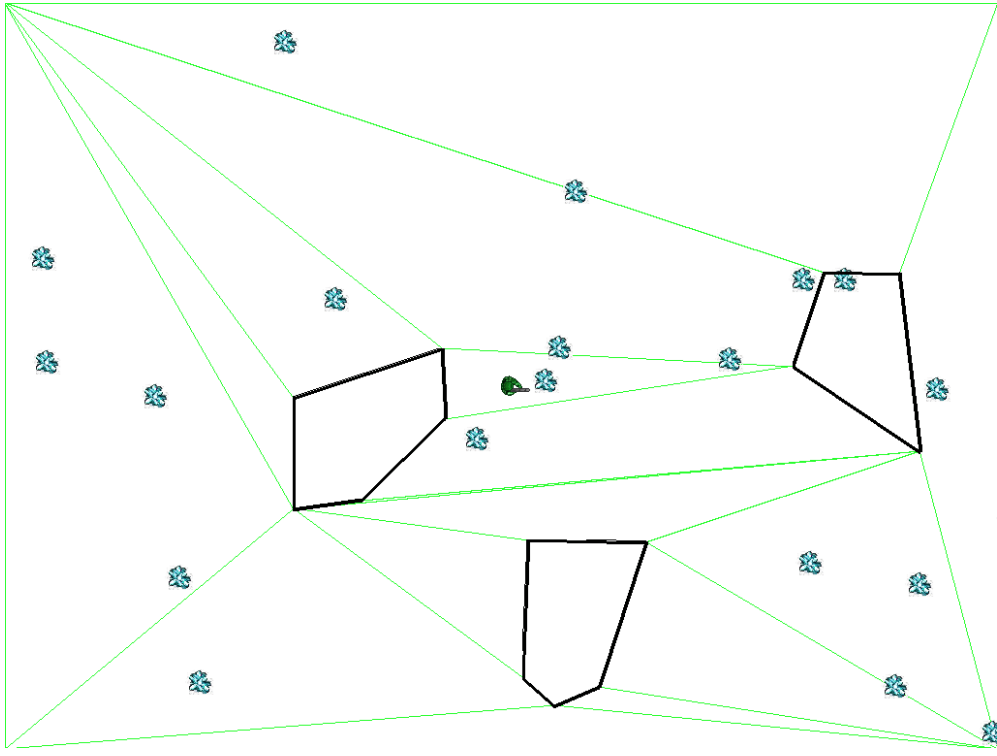
M4: NavMesh

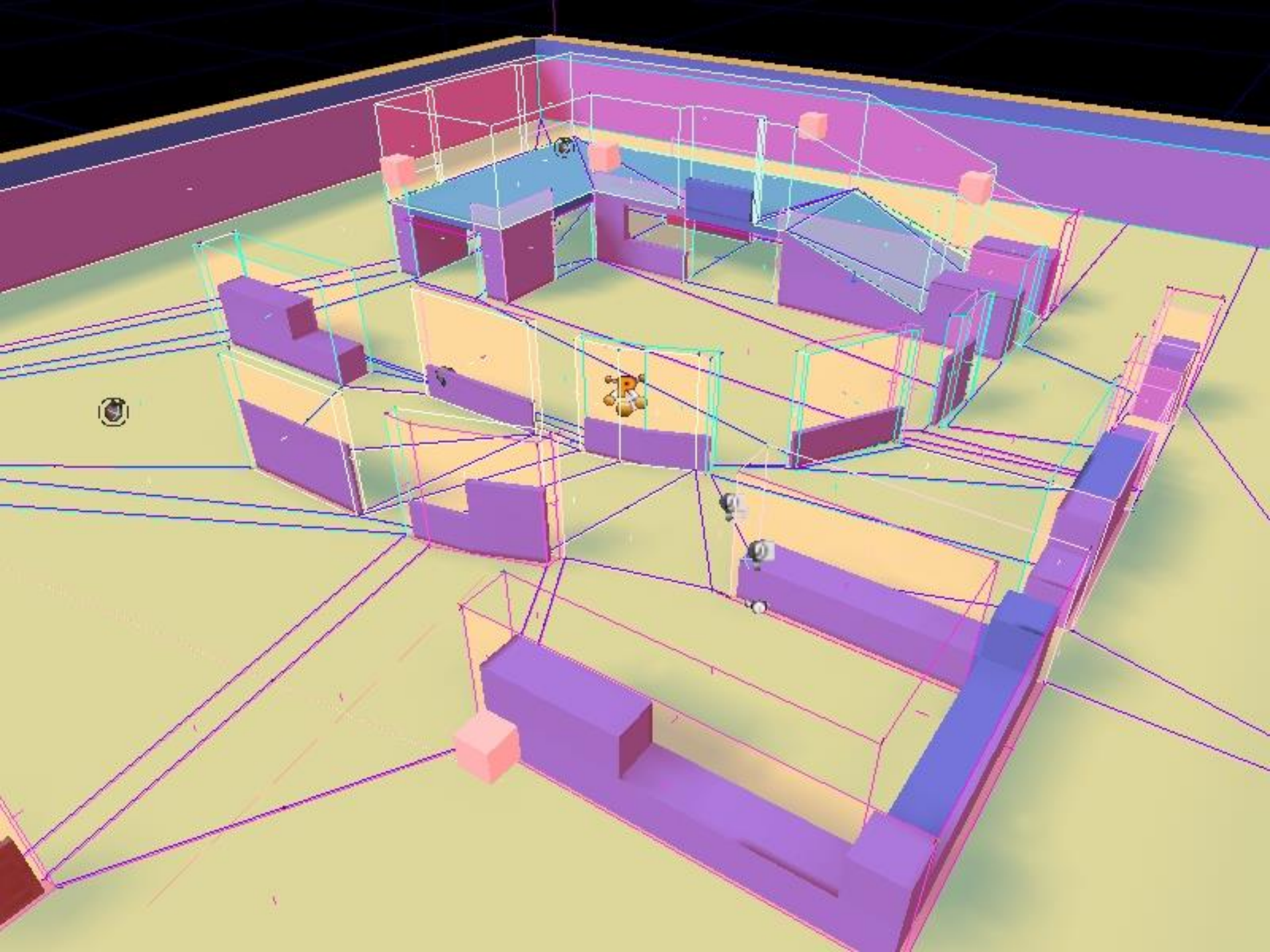
- Win: compact rep, fast search, auto create
- Each node (edges) is a convex polygon
- Convex = Any point within the polygon is unobstructed from any other
- Can be generated from the polygons used to define a map

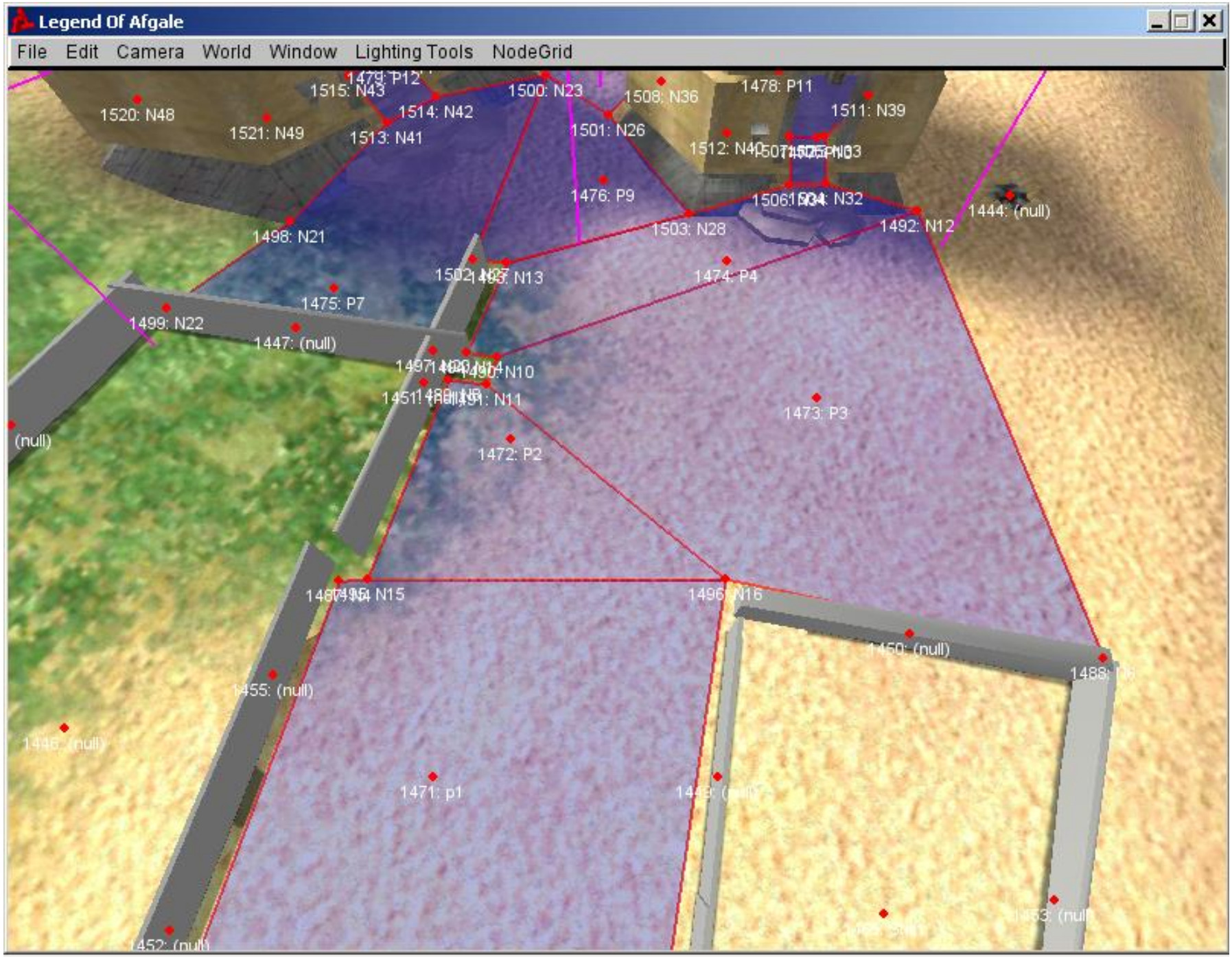


Generating the Mesh

- Lots of algorithms
- Optimal:
  - Fewest polygons, smallest discretization possible
  - NP-complete
- Greedy
  - Find triangles guarantees convex
  - Merge triangles

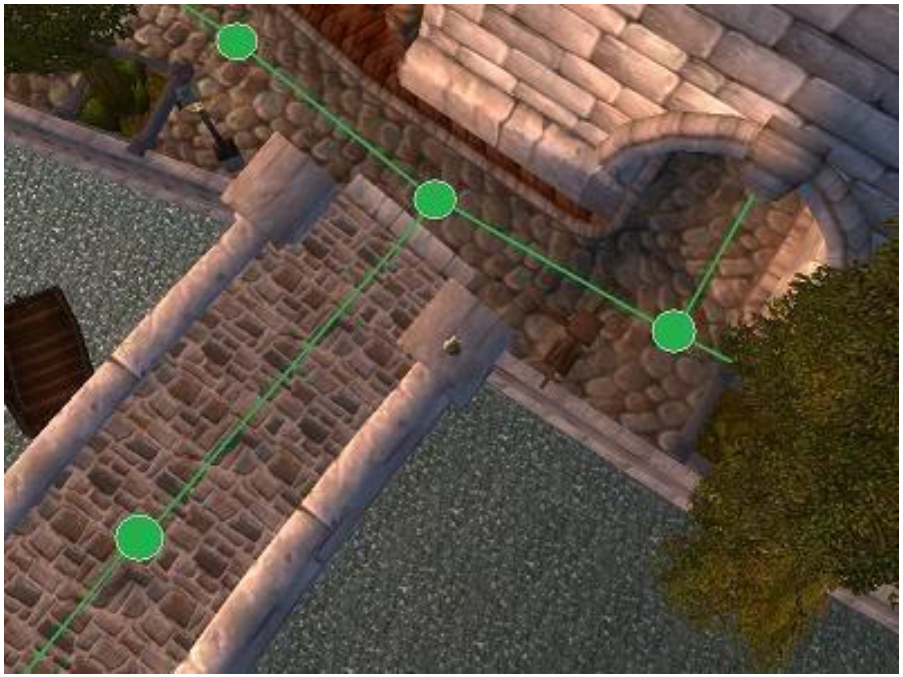


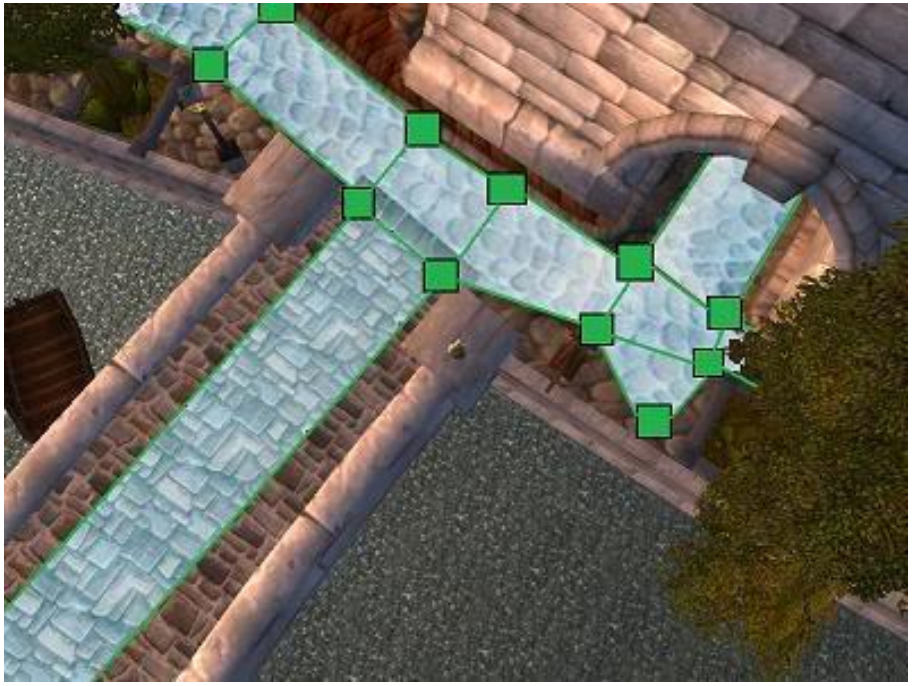



Legend Of Afgale
File Edit Camera World Window Lighting Tools NodeGrid
1520: N48
1521: N49
1515: N43
1514: N42
1513: N41
1500: N23
1508: N36
1478: P11
1511: N39
1501: N26
1512: N40
1476: P9
1498: N21
1503: N28
1492: N12
1444: (null)
1474: P4
1475: P7
1499: N22
1447: (null)
1473: P3
1472: P2
1455: (null)
1471: p1
1450: (null)
1452: (null)

Waypoints vs. NavMesh

5 Reasons why waypoints fall short





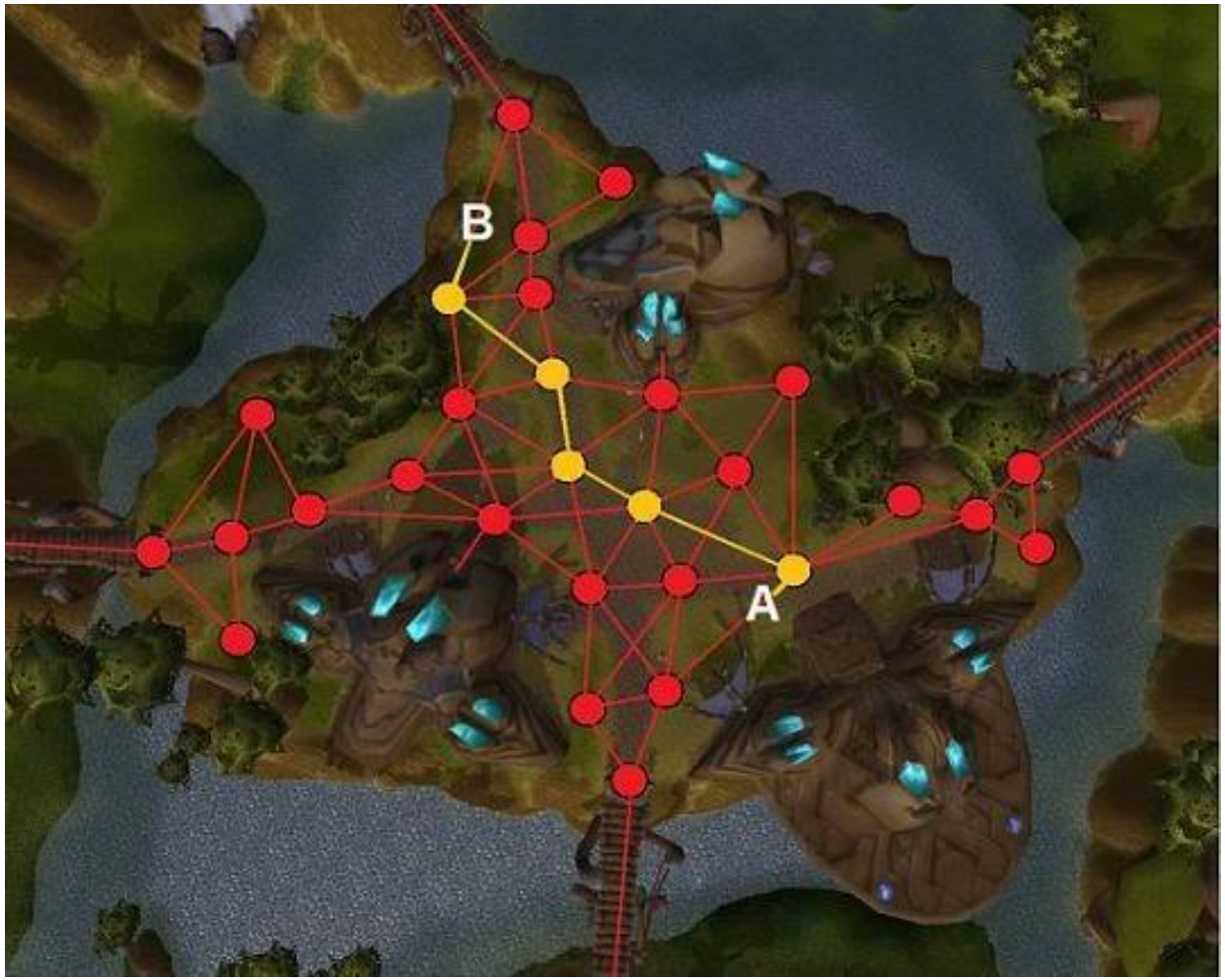
1) Some worlds need WAY too many

Waypoints. Need more?

NavMesh

2) Waypoints make NPCs zig-zag

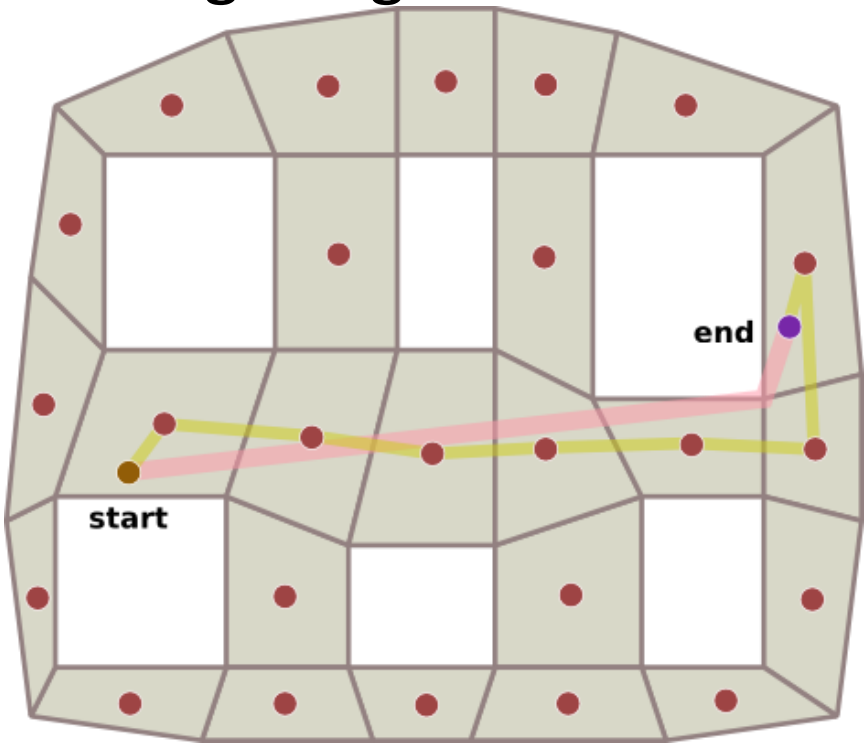
NPC path



NavMesh path

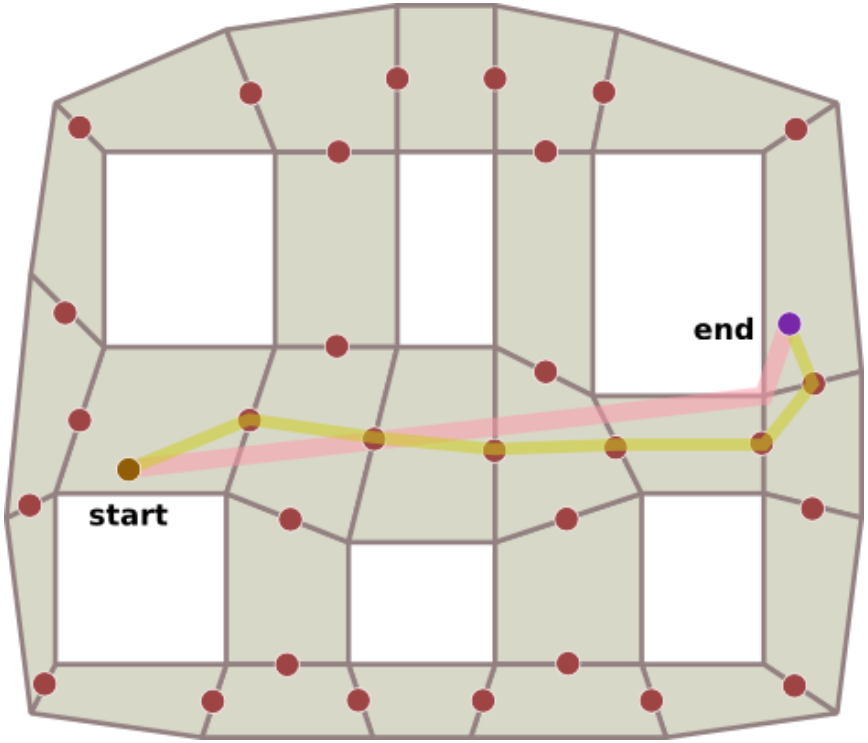
Nav Meshes + Waypoints

- Put a waypoint in center of each nav mesh
  - It's important to get a good set of nav meshes



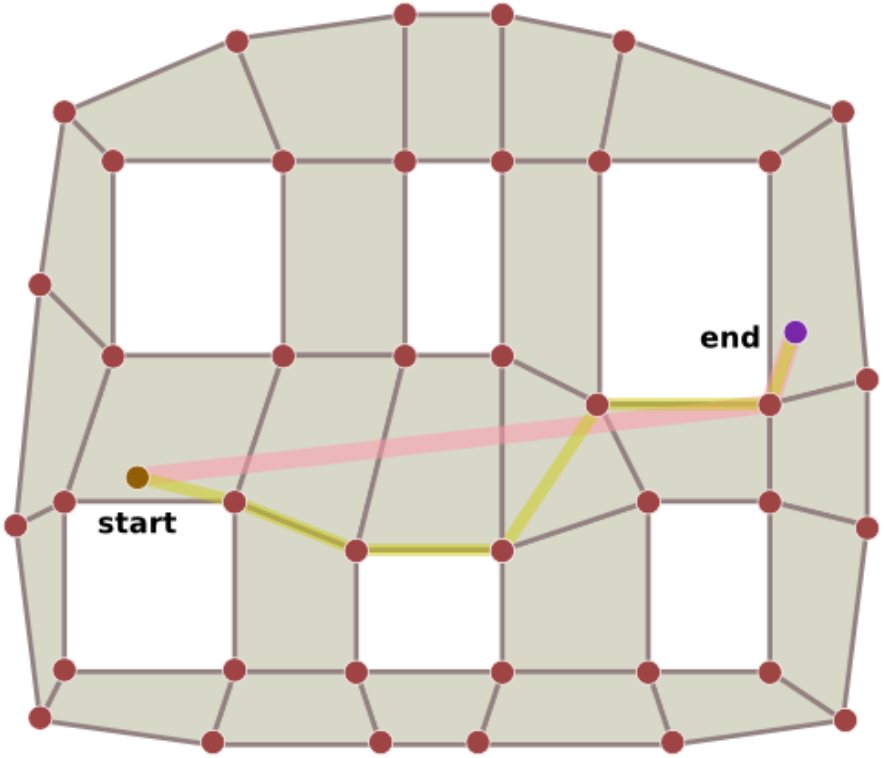
Nav Meshes + Waypoints

- Put a waypoint at adjoining edges



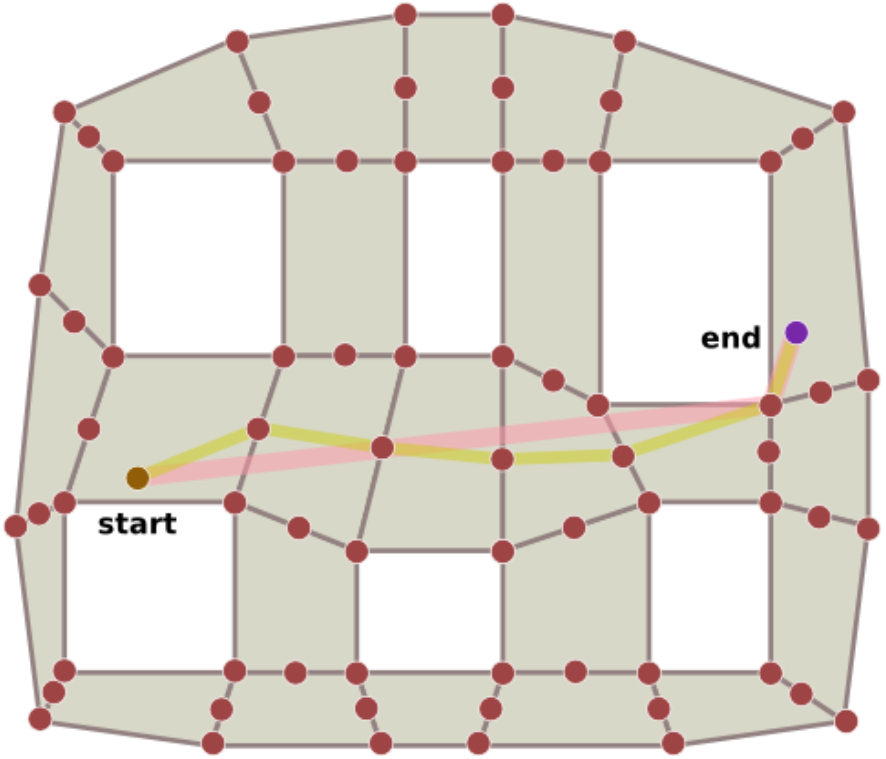
Nav Meshes + Waypoints

- Put a waypoint at corners of obstacles



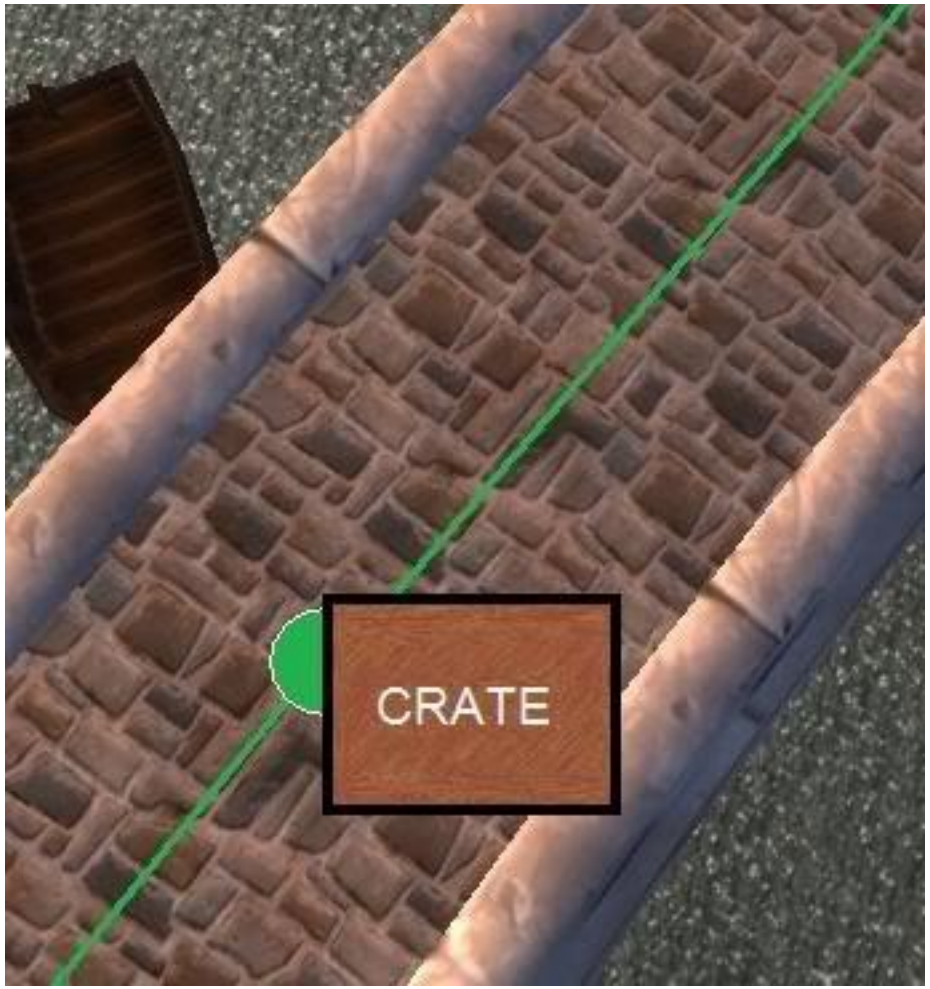
Nav Meshes + Waypoints

- Put a waypoint at edges and corners

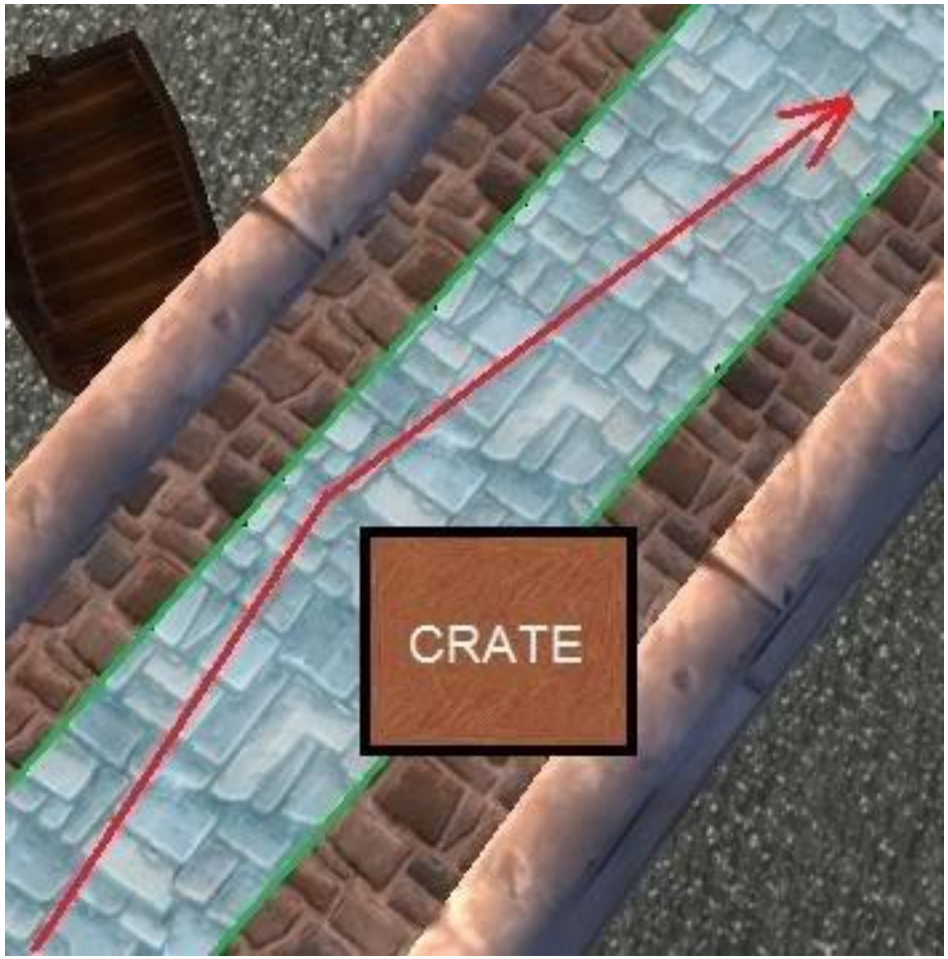


NavMesh Smoothing

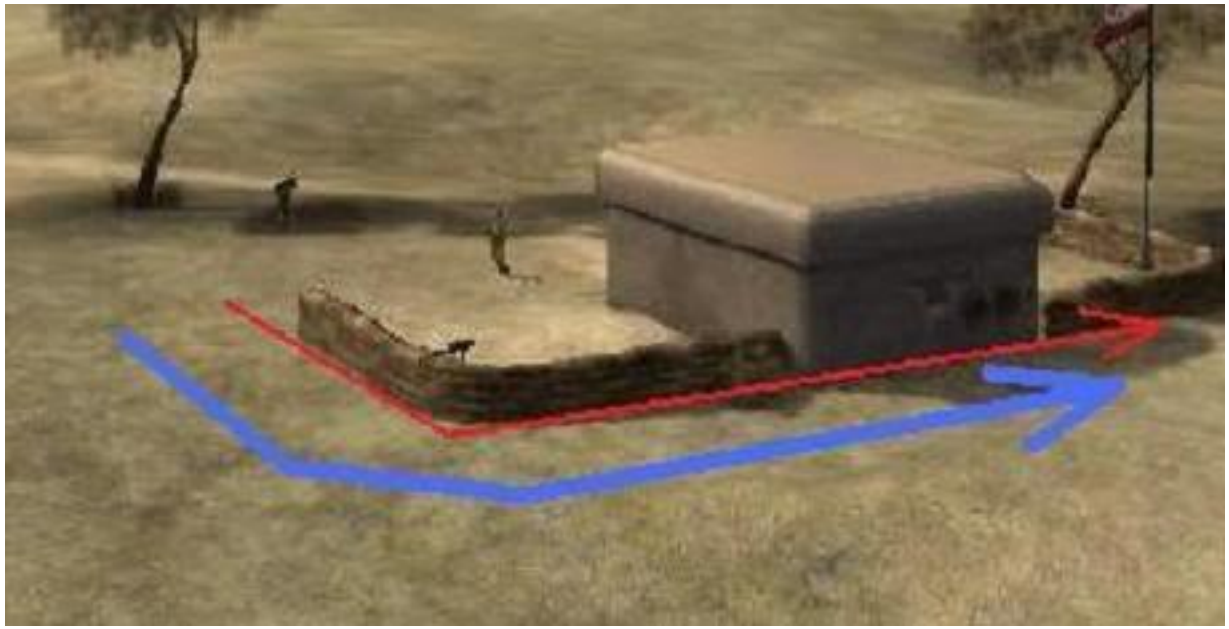
3) Waypoints don’t allow for path correction



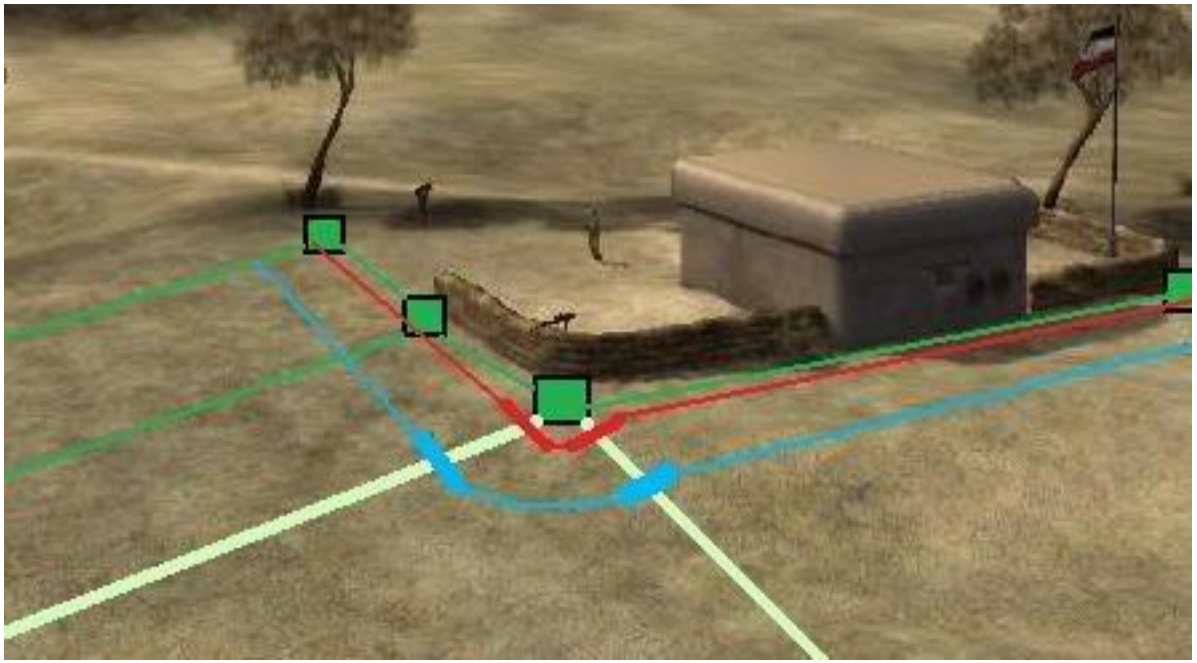
NavMesh solution



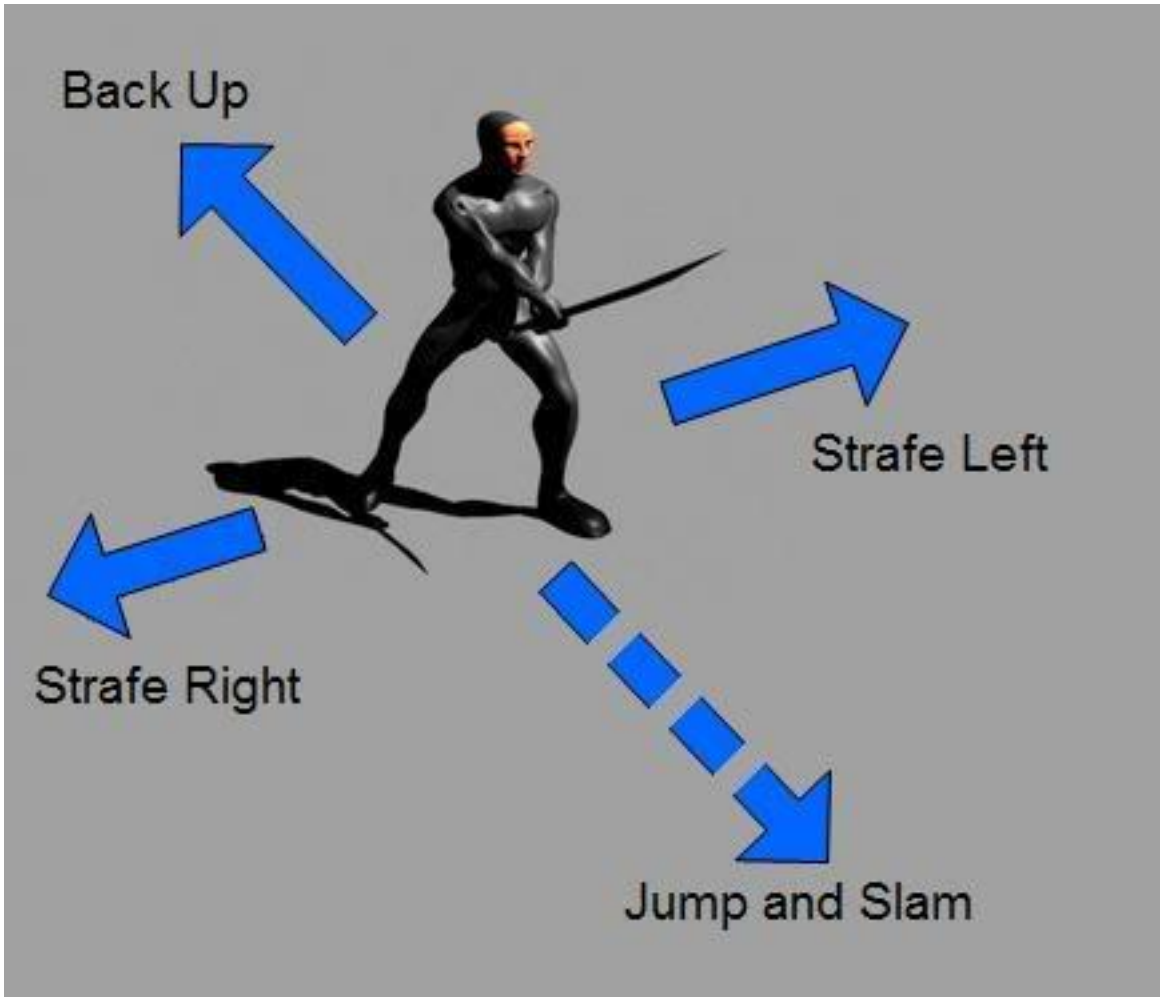
4) Waypoints don’t work well for different characters



NavMesh solution



5) They don’t hold enough data



NavMesh

- Isn’t pathfinding on a NavMesh slower?
  - No

- Graph usually has fewer nodes
- Makes the assumption that you can locally get from node (edge) to node (edge) because of the convex polygon construction

NavMesh

- Don’t they take up a lot of memory?
  - No

- Can be smaller than dense waypoint graphs
- Smaller than collision mesh (ignores walls, etc.)
- Fairly compact representation

FPS Example

- Let’s see that again (waypoints…)
  - http://www.youtube.com/watch?v=WzYEZVI46Uw
- Cheating / hiding the problem
  - Most AIs don't live long enough to let you spot the flaws in their pathfinding (LOS stop, shoot)
  - Many 1P FPS, AIs don't move very much, shoot from relatively fixed position.
  - FPS games with AI sidekicks kill the enemy AIs so quickly they don't have time to move very far.

FPS implications

- What if we force characters to use melee weapons (e.g. Covenant soldiers in Halo, the Icarus stealth assassins in F.E.A.R., or the Metroids in a Metroid Prime game)? Which technique did they use?

Designers need to be able to add info…

How do you handle walking under bridges?

Is this good for all games?

- Not necessarily
- Find the right solution for your problem