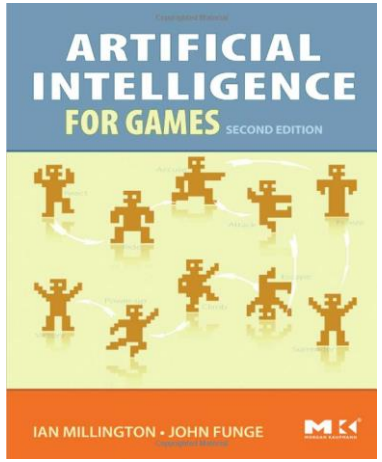
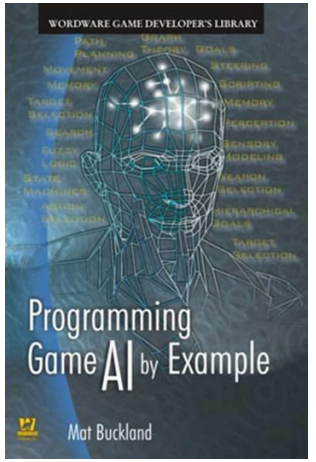




"the question of whether computers can think is like the question of whether submarines can swim" -- Dijkstra

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

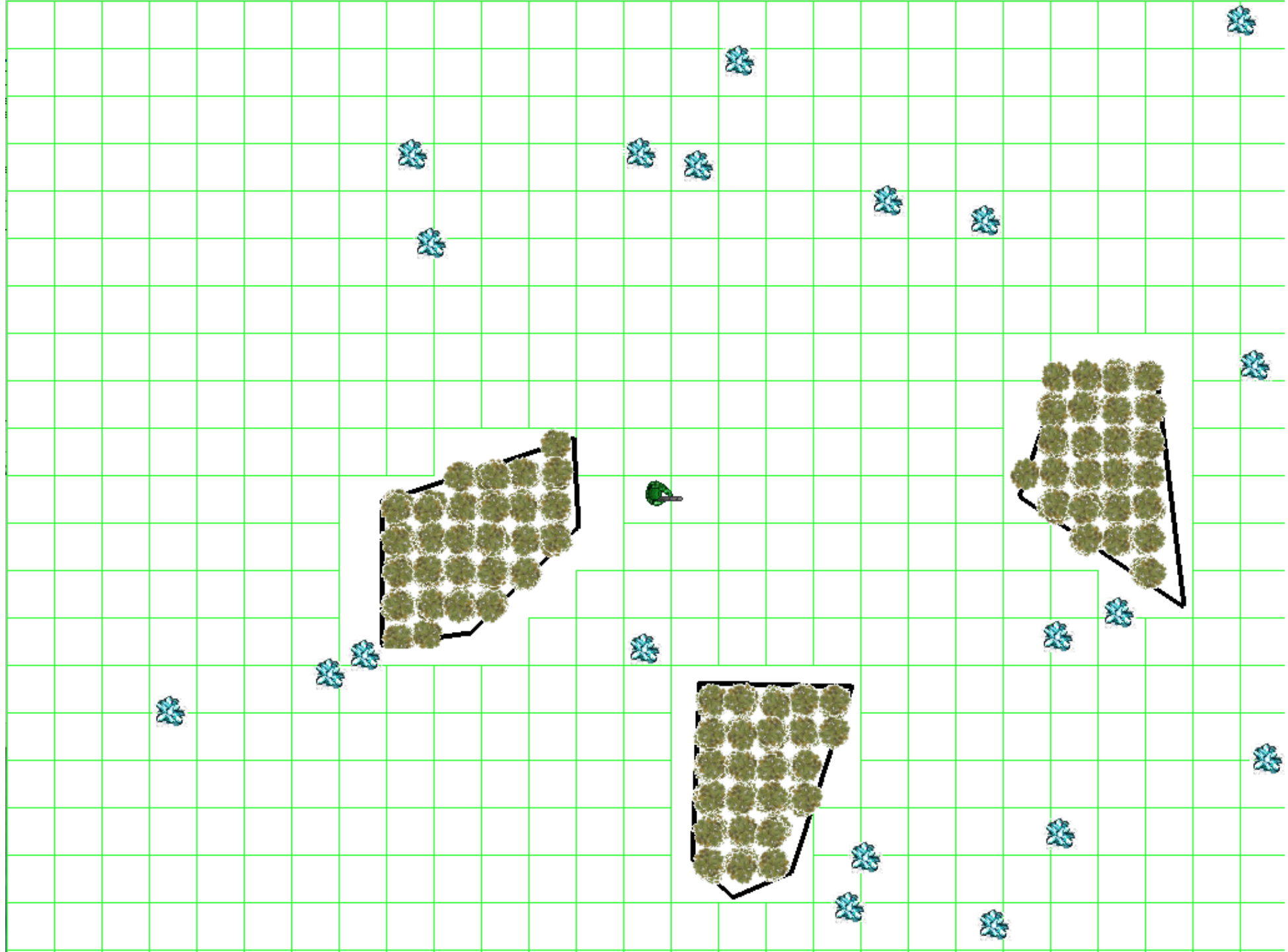
_____: A rule of thumb, simplification, or educated guess that reduces or limits the search for solutions in domains that are difficult and poorly understood.

Announcements

- HW1 due Sunday night, January 21 @ 11:55pm
- HW2 is much more challenging than HW1. Start early!
- Waitlist? Attendance
- Game engine & HWs – piazza only; please no posting
- Office hours
  - https://piazza.com/gatech/spring2018/cs4731cs7632/staff
- Special lectures (Guzdial, Russell)

Grid Generation Hints

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



PREVIOUSLY ON…

Class N-2: What is…

- GAI?
  - Set of tricks and techniques to bring about a particular game design.
  - Goals include:
    - enhancing the player's engagement, enjoyment, and experience
      - End behavior is the target
      - Do better than random
    - doing things the player or designer cannot do or don't want to do
      - replace real people when they are unwilling or unavailable to play
      - aid for designers and developers
    - making the entities, opponents, agents, companions, etc. in games appear intelligent
    - believable characters / looking convincing
- A Game?
  - A system of rules and a goal and agency.

N-2: How/Why distinct from “academic AI”

- Supporting the player experience
- Good game AI == matching right behaviors to right algorithms
- Product is the target, not clever coding – ends justify means. FUN
- Illusion of intelligence
- “Magic Circle” (Rules of play: game design fundamentals)
- Elegance in simplicity & the complexity fallacy
- Quality control & resource limits
- Fun vs smart: goal is not always to beat the player
- Optimal/rational is rarely the right thing to do

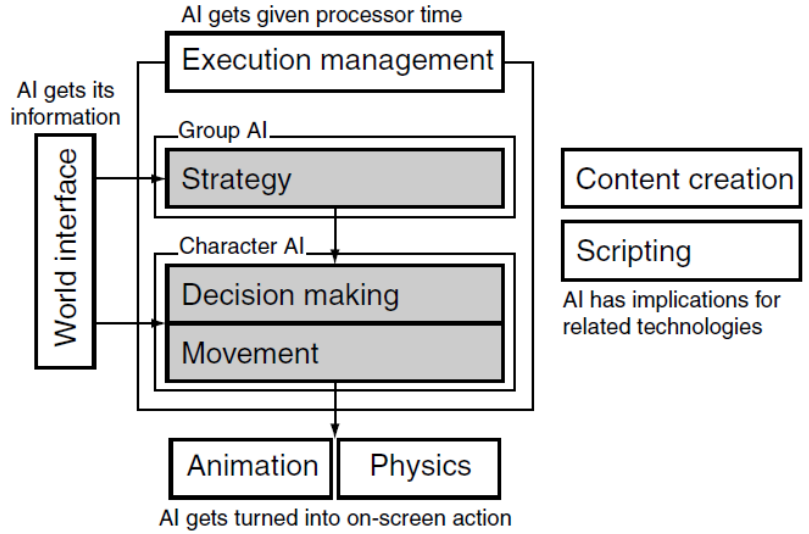
N-2: Common (game) “AI” Tricks?

- Move before firing – no cheap shots
- Be visible
- Have horrible aim (being Rambo is fun)
- Miss the first time
- Warn the player
- Attack “kung fu” style (Fist of Fury; BL vs School)
- Tell the player what you are doing (especially companions)
- React to own mistakes
- Pull back at the last minute
- Intentional vulnerabilities or predictable patterns

Liden, “Artificial Stupidity: The Art of Intentional Mistakes”. *AI Game Programming Wisdom*.

N-2: Major ways GameAI is used…

- In game
  - Movement
  - Decision making
  - Strategy
  - Tailoring/adapting to player individual differences
  - Drama Management
- Out of game
  - PCG
  - Quality control / testing



M&F Fig 1.1

N-2: Why AI is important for games

- Why is it essential to the modeled world?
  - NPC's of all types: opponents, helpers, extras, …
- How can it hurt?
  - Unrealistic characters → reduced immersion
  - Stupid, lame behaviors → reduced fun
  - Superhuman behaviors → reduced fun
- Until recently, given short shrift by developers. Why?
  - Graphics ate almost all the resources
  - Can't start developing the AI until the modeled world was ready to run
    - AI development always late in development cycle
- Situation rapidly changing / changed. How?
  - AI now viewed as helpful in selling the product
  - Still one of the key constraints on game design

Credit: Dr. Ken Forbus

N-1: Intro, Graph and Search

1. How do intentional mistakes help games?
2. What defines a graph?
3. What defines graph search?
4. Name 3 uniformed graphs search algorithms.
5. Name an informed graph search algorithm?
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

Graphs: Killer App in GAI

- Navigation / Pathfinding
- Navgraph: abstraction of all locations and their connections
- Cost / weight can represent terrain features (water, mud, hill), stealth (sound to traverse), etc
- What to do when …
  - Map features move
  - Map is continuous, or 100K+ nodes?
  - 3D spaces?

Graph Search

- Uninformed (all nodes are same)
  - DFS (stack – lifo), BFS (queue – fifo)
  - Iterative-deepening (Depth-limited)
- Informed (pick order of node expansion)
  - Dijkstra – guarantee shortest path ($E\log_2 N$)
  - A* (IDA*)…. Dijkstra + heuristic
  - D*
- Hierarchical can help

http://en.wikipedia.org/wiki/A*_search_algorithm

Heuristics

- [dictionary]*"A rule of thumb, simplification, or educated guess that reduces or limits the search for solutions in domains that are difficult and poorly understood."*
- *h(n)* = estimated cost of cheapest path from n to goal (with goal == 0)

Path finding models

1. **Tile-based graph – “grid navigation”**
   - Simplest topography
   - assume static obstacles
   - imaginary latice of cells superimposed over an environment such that an agent can be in one cell at a time.
   - Moving in a grid is relatively straightforward: from any cell, an agent can traverse to any of its four (or eight) neighboring cells
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* (or perhaps greedy), Dijkstra
  - http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html

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

Question

What are pros and cons of a grid representation of space in terms of character movement?

Grid navigation: pros

- Discrete space is simple
- Can be generated algorithmically at runtime (Hw1)
- Good for large number of units
- A*/greedy search works really well on grids (uniform action cost, not many tricky spots)

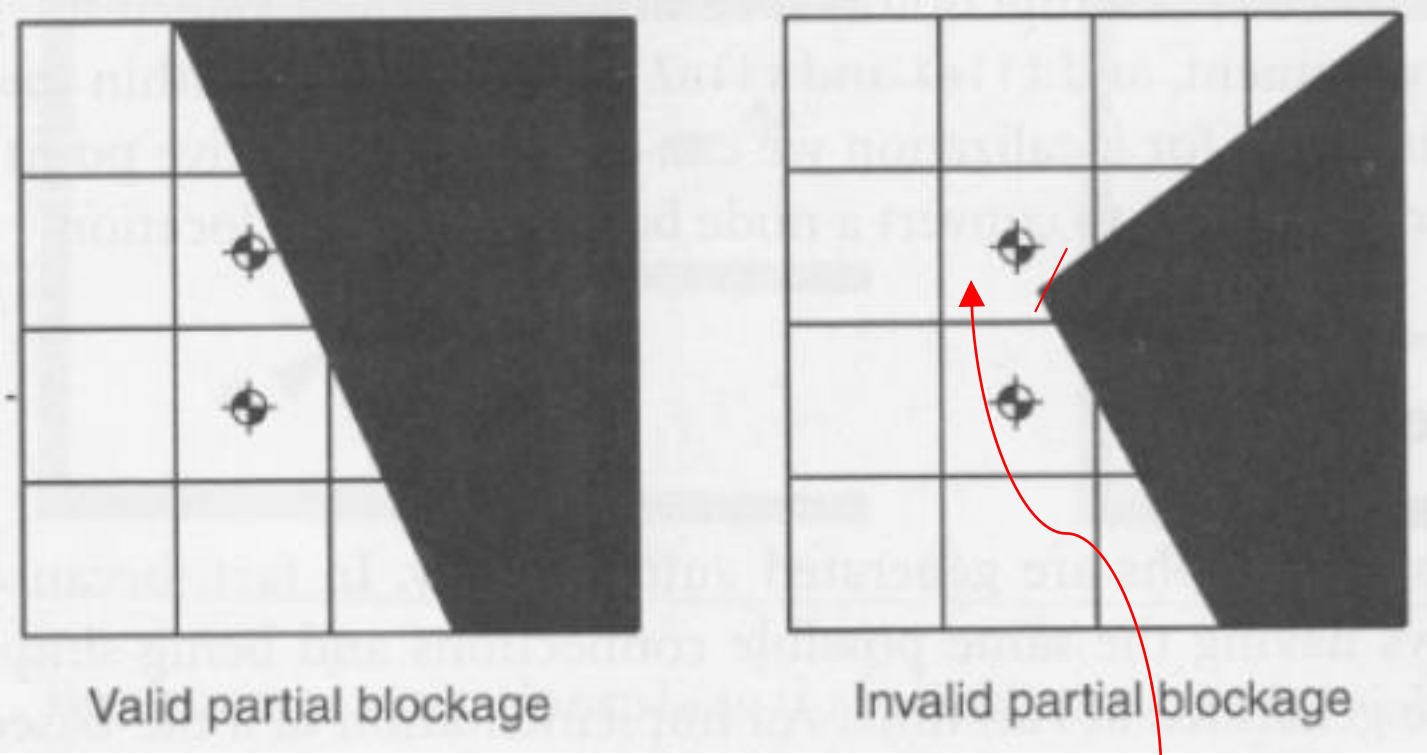
Grid navigation: cons

- 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. 100x100 map == 10k nodes and ~78k edges

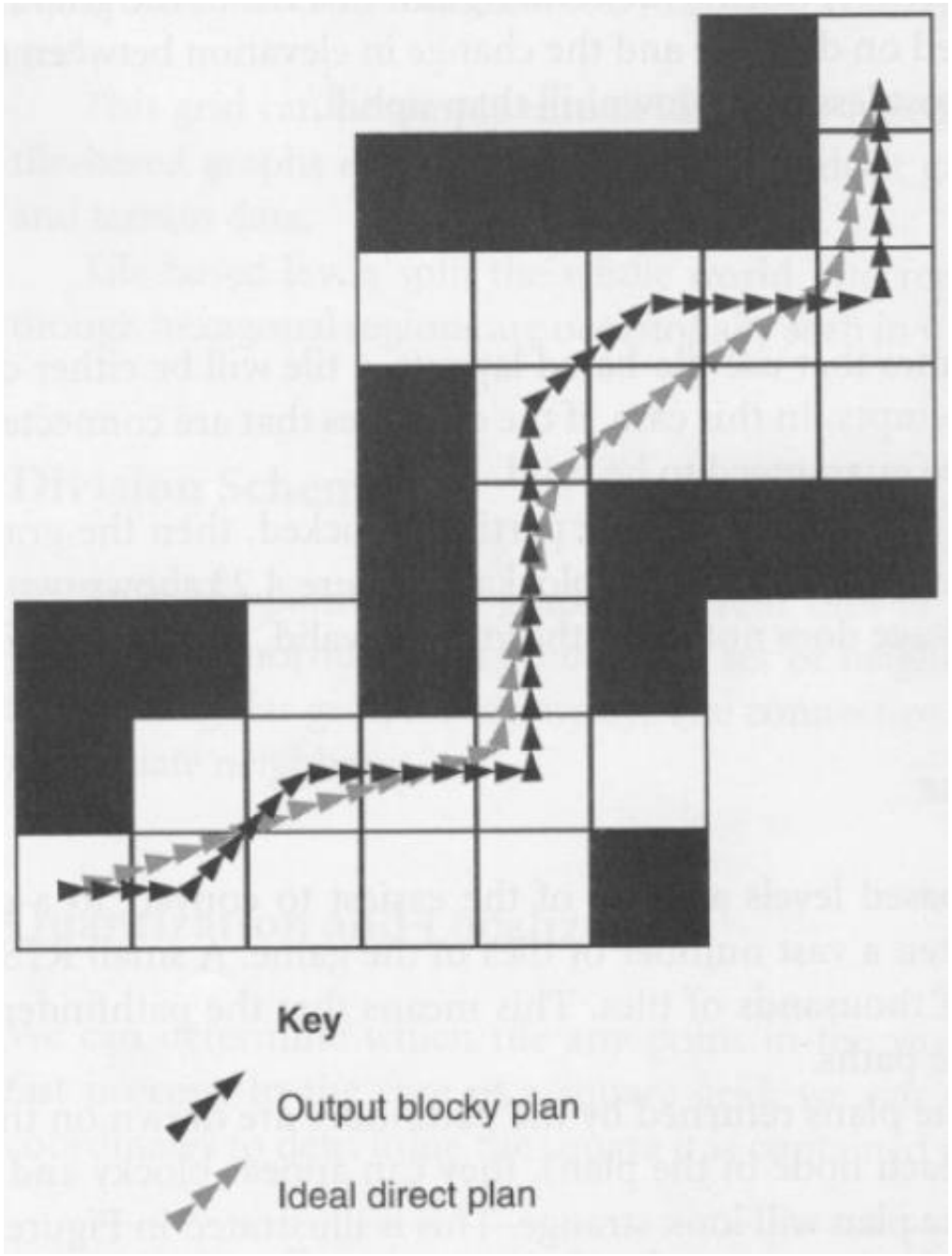
New Problems

- 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)
- Awkward agent movement
- Long search times

Validity



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



Fixing awkward agent movement:

- String pulling
- Splines
- Hierarchical A*

Graphs, Search, & Path Planning
Continued: Models of world for path planning

2018-01-16;

See also: Buckland Ch 5 & 8,

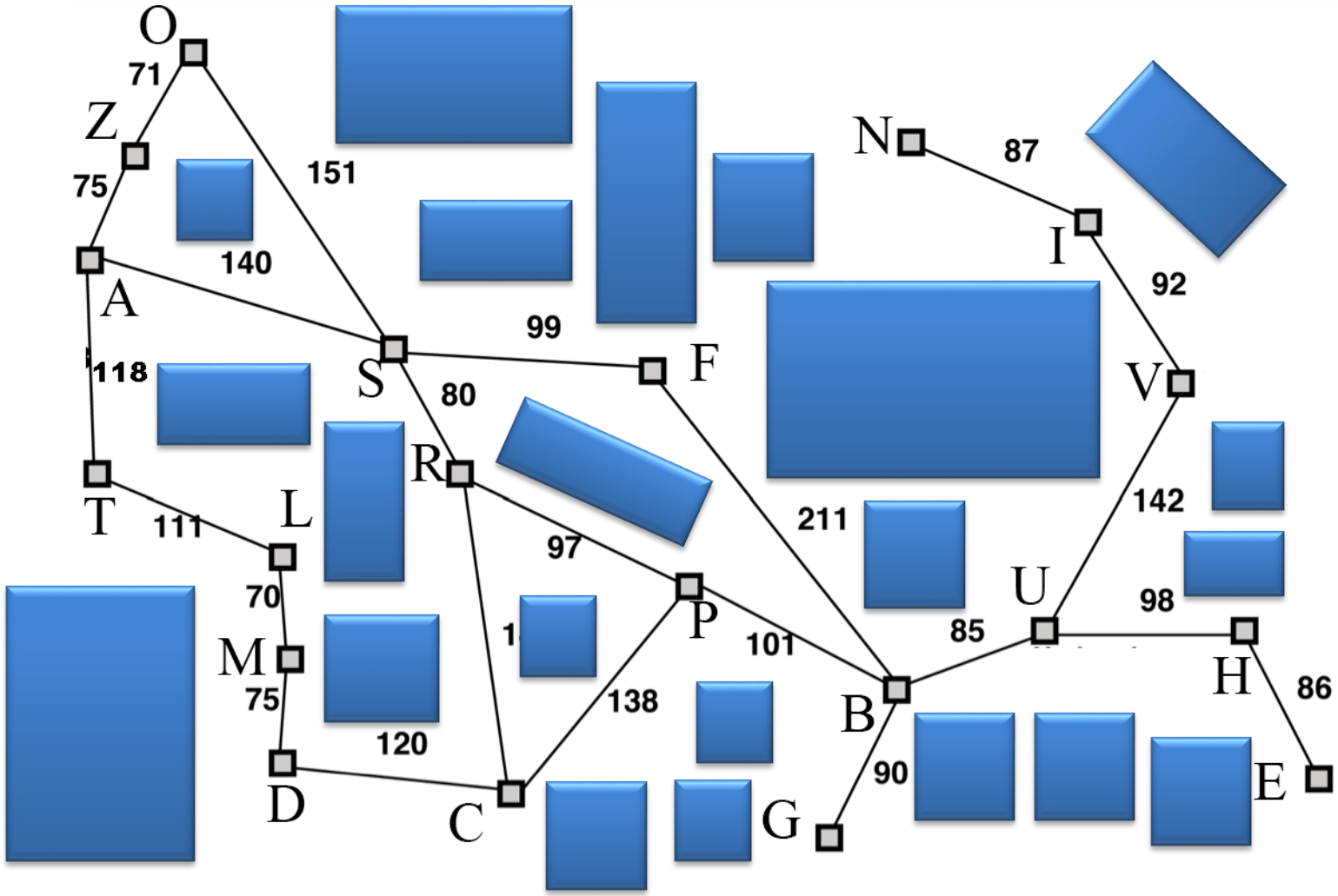
Millington & Funge Ch 4

Path finding models

1. Tile-based graph – “grid navigation”
2. **Path Networks / Points of Visibility NavGraph**
   - does not require the agent to be at one of the path nodes at all times. The agent can be at any point in the terrain.
   - When the agent needs to move to a different location and an obstacle is in the way, the agent can move to the nearest path node accessible by straight-line movement and then find a path through the edges of the path network to another path node near to the desired destination.
3. Expanded Geometry
4. NavMesh

Model 2: Path Networks

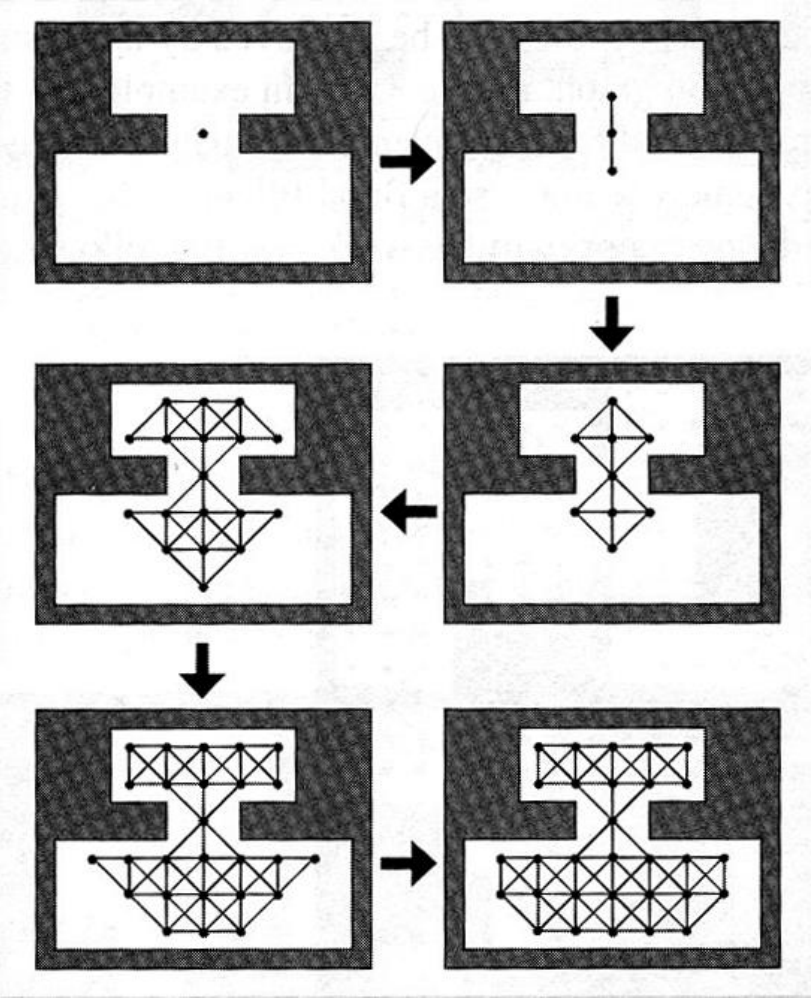
- POV: Points of visibility NavGraph (see B CH 8)
- Discretization of space into sparse network of nodes
- Two-tiered navigation system
  - Local, continuous
  - Remote
- Connects points *visible to each other* in all important areas of map
- Usually hand-tailored (can use flood-fill)


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

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)

Flood Fill



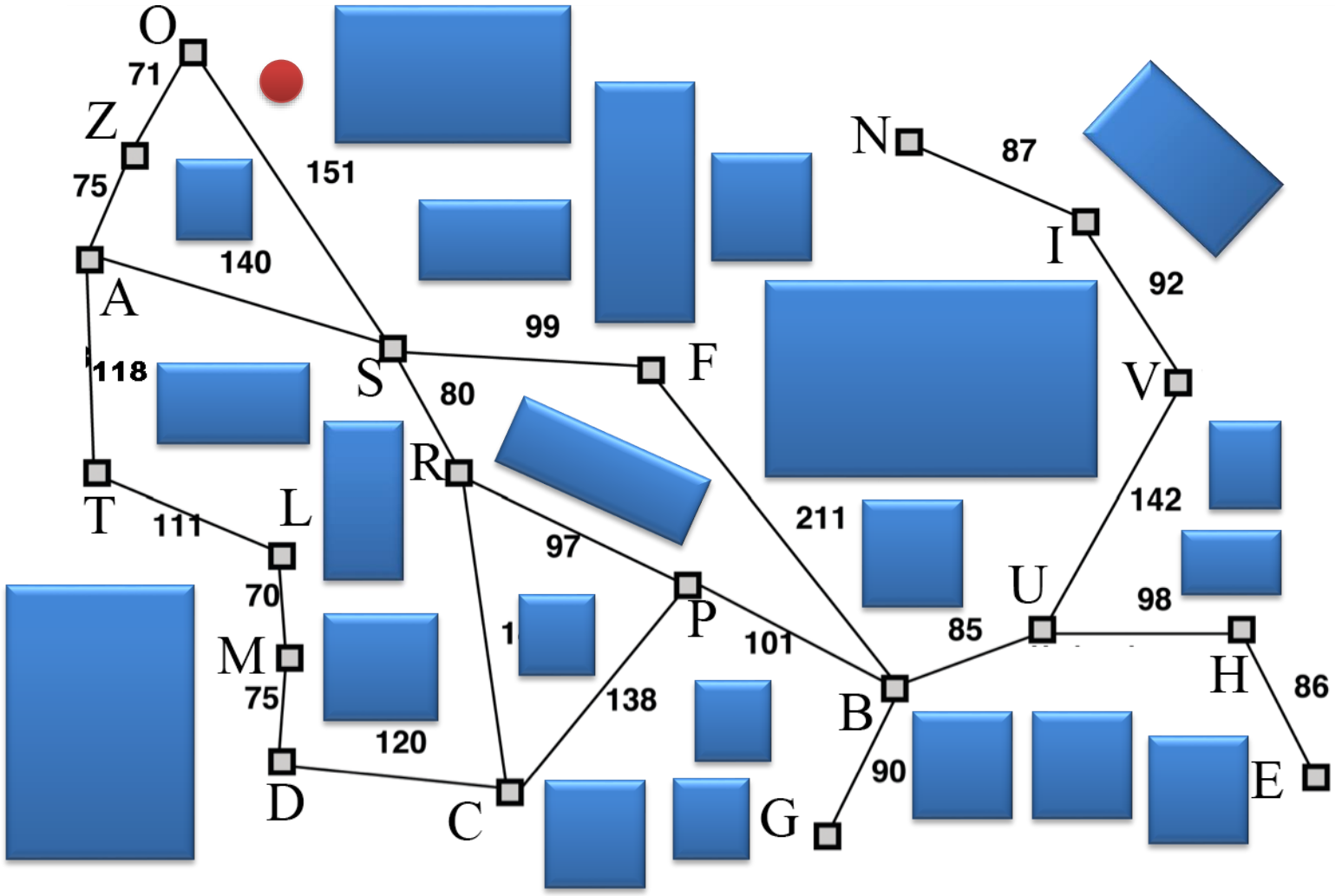
- Start with “seed”
- “grow” graph
- Designer can move, delete, or add nodes
- Ensure **all nodes and edges are at least as far from walls as agents bounding radius**

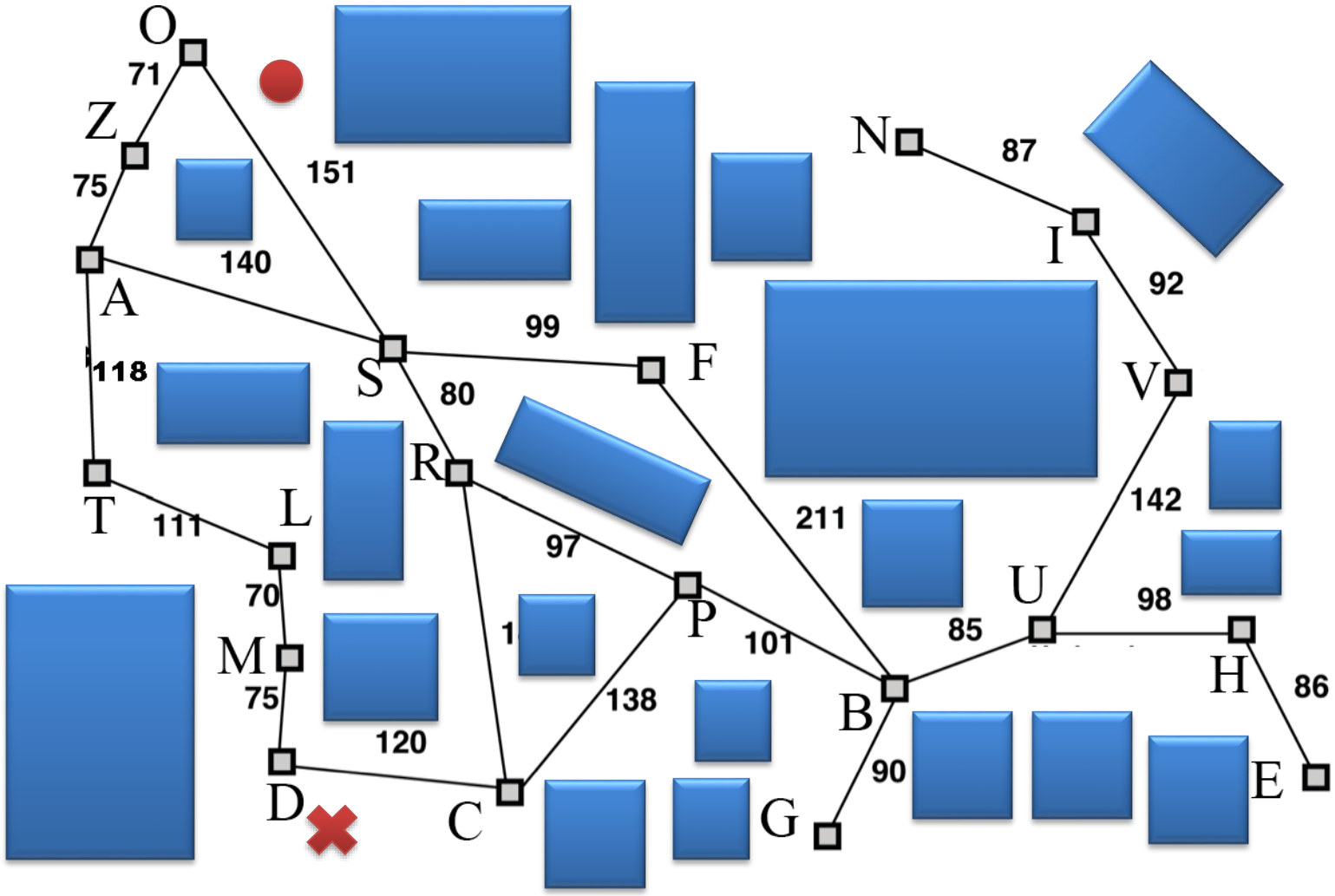
Buckland Figure 8.7

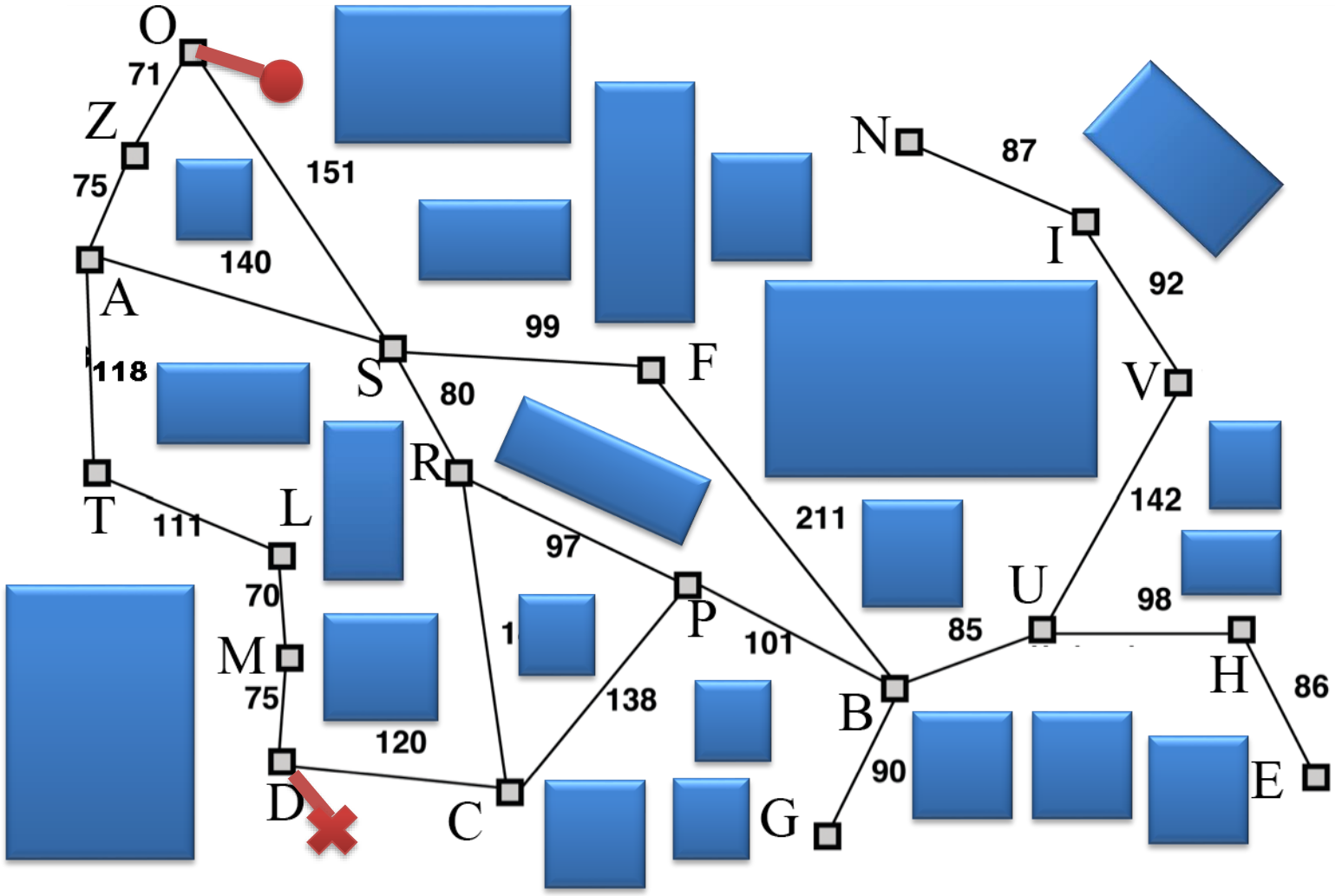
Using the path network

- 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

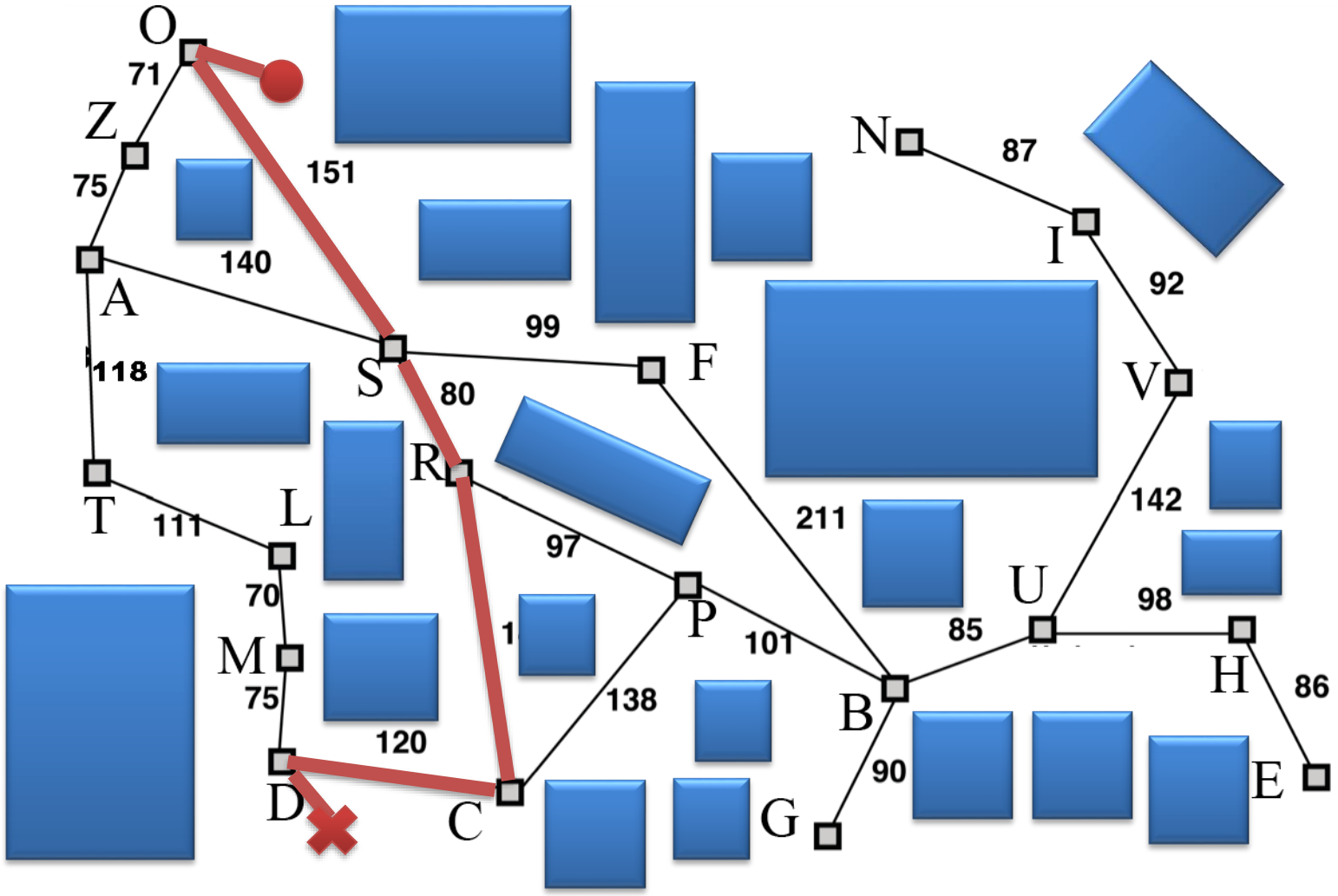
Problem: Unsightly paths.


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

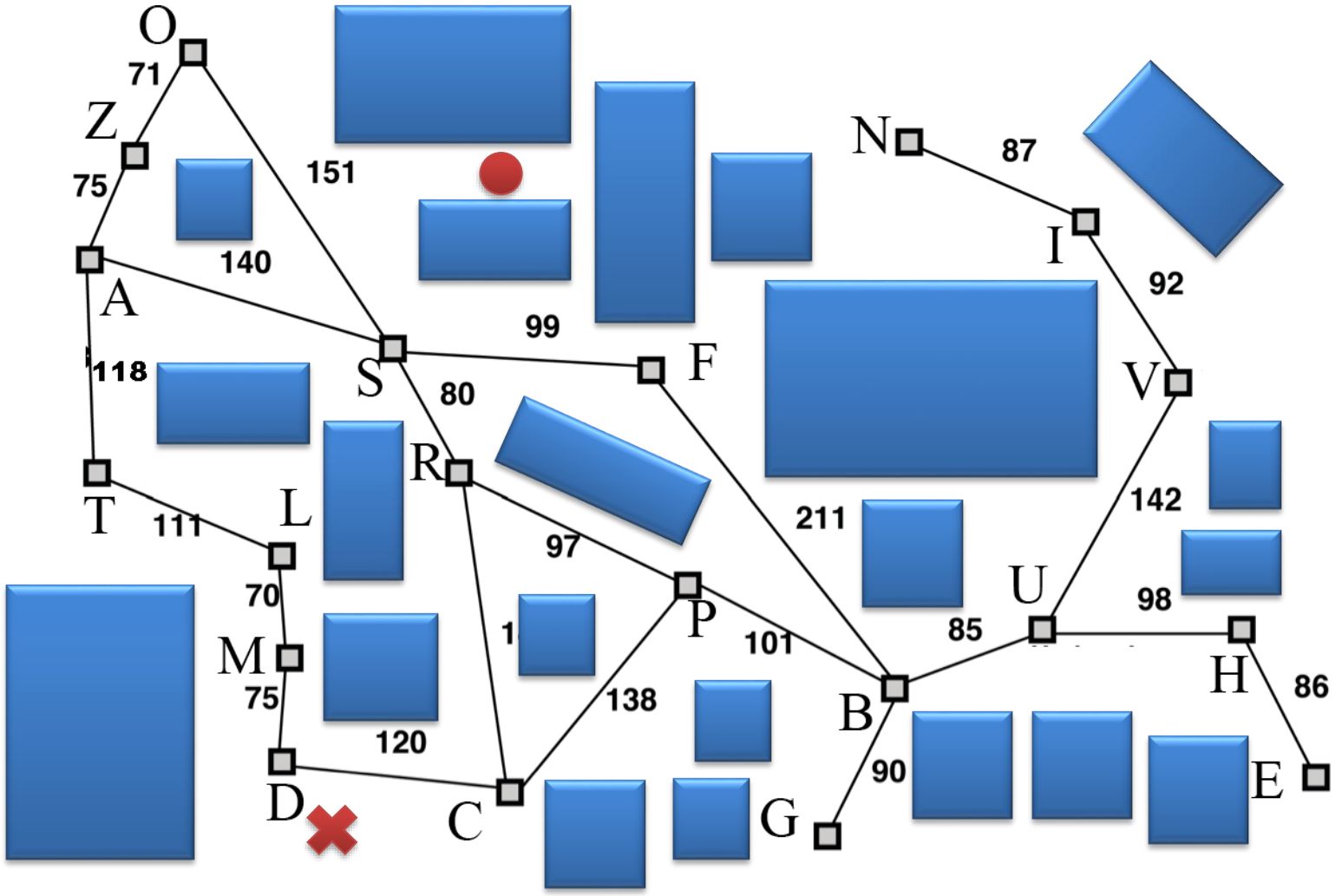

O
71
Z
75
151
140
A
99
118
S
80
F
N
87
I
92
V
R
T
111
L
97
211
142
70
P
101
U
85
98
M
75
120
138
B
H
86
D
C
G
90
E


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

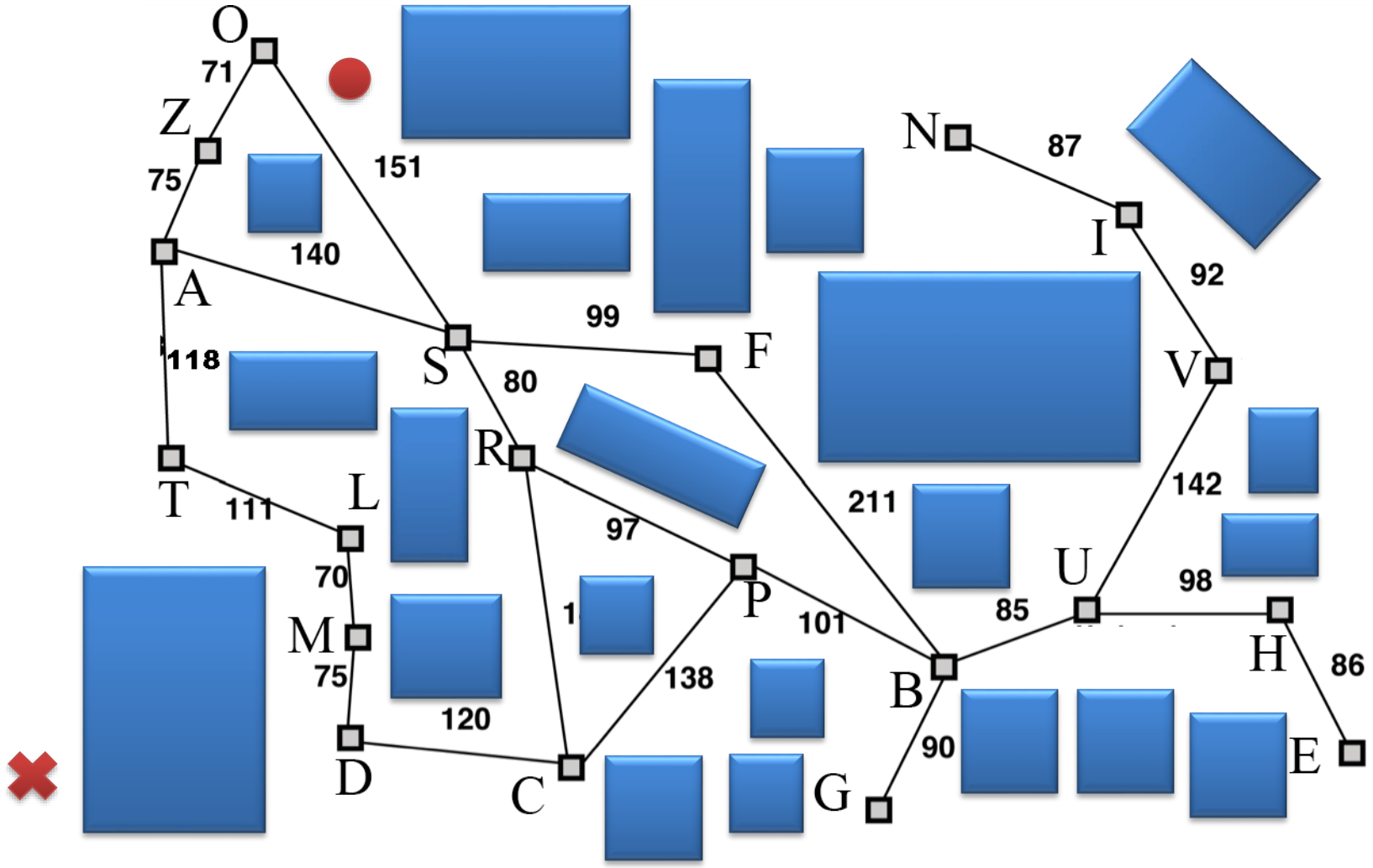
Unnecessary double-backs (unsightly)



Source not in sight of navigation point

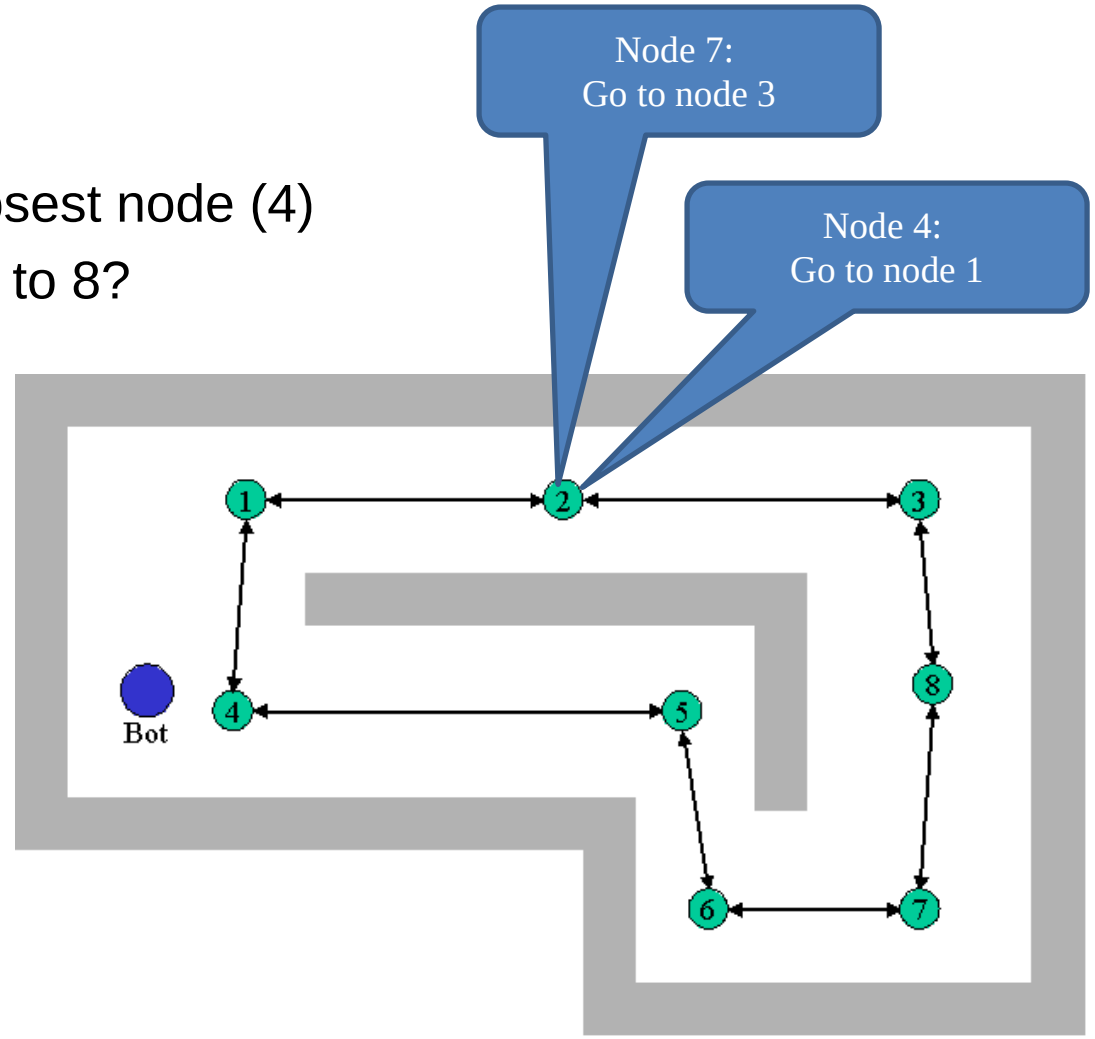


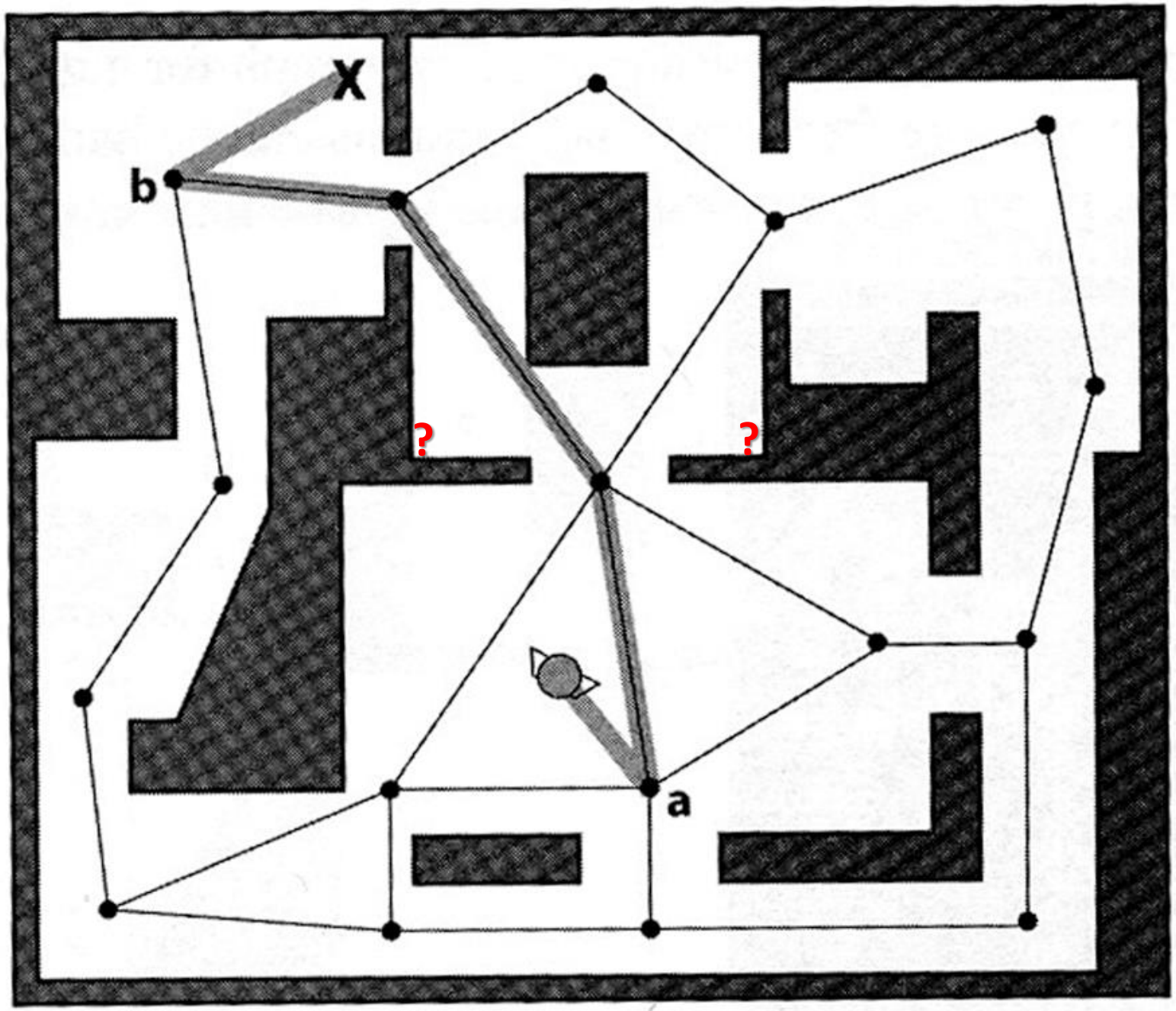
Destination not in sight of a navigation point

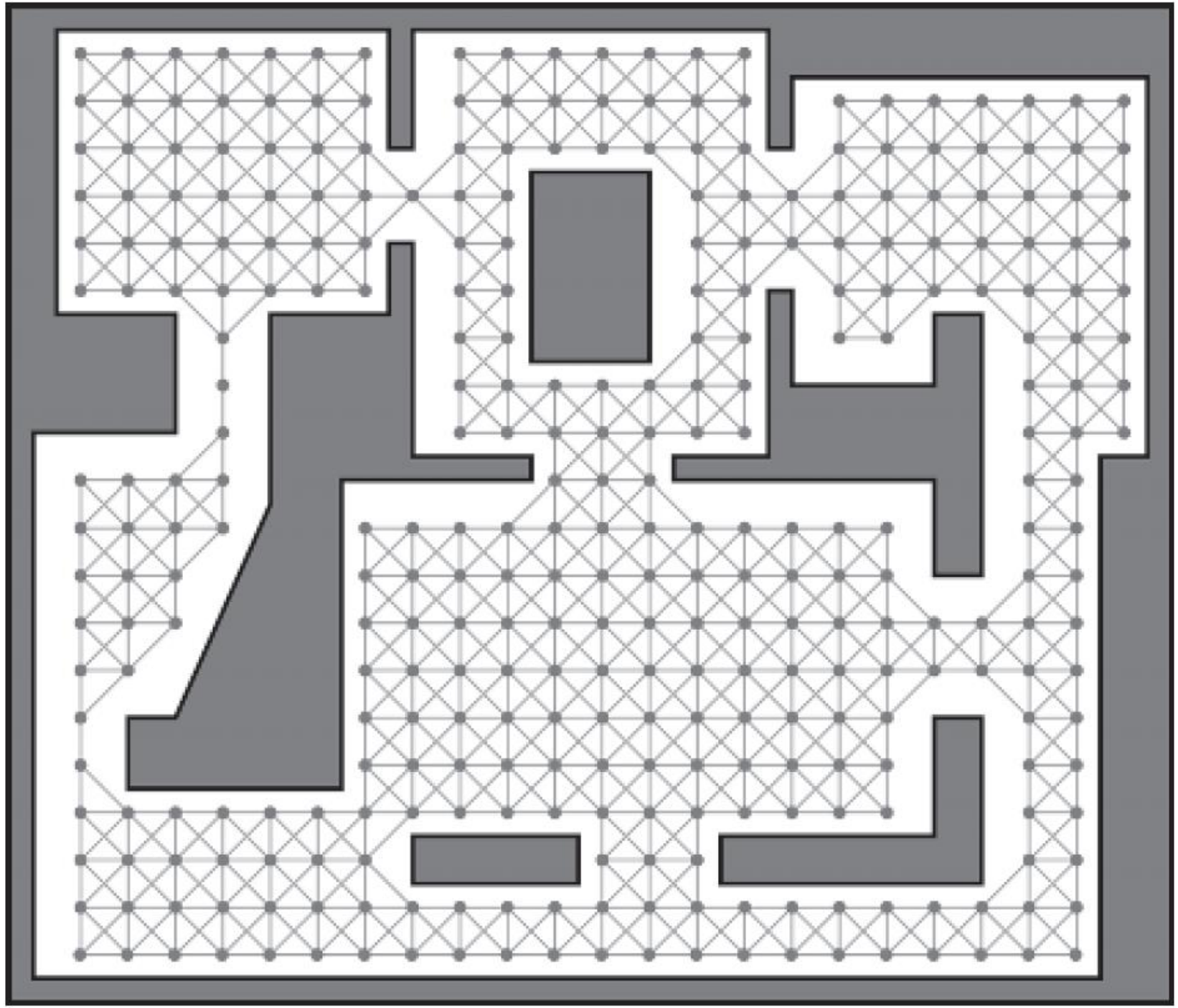


Using a path node network

- Bot decides to go to 8
- Get on the network at closest node (4)
- Ask: where to next to get to 8?
- Global table, or store in nodes themselves




X
b
?
?
a



Buckland Figure 8.6

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: $O(|V|^2)$
- Fully path node process: $O(|V|^3)$

* Can run in $O(|E| + |V|\log|V|)$

* Run Djikstra |V| times.

Question

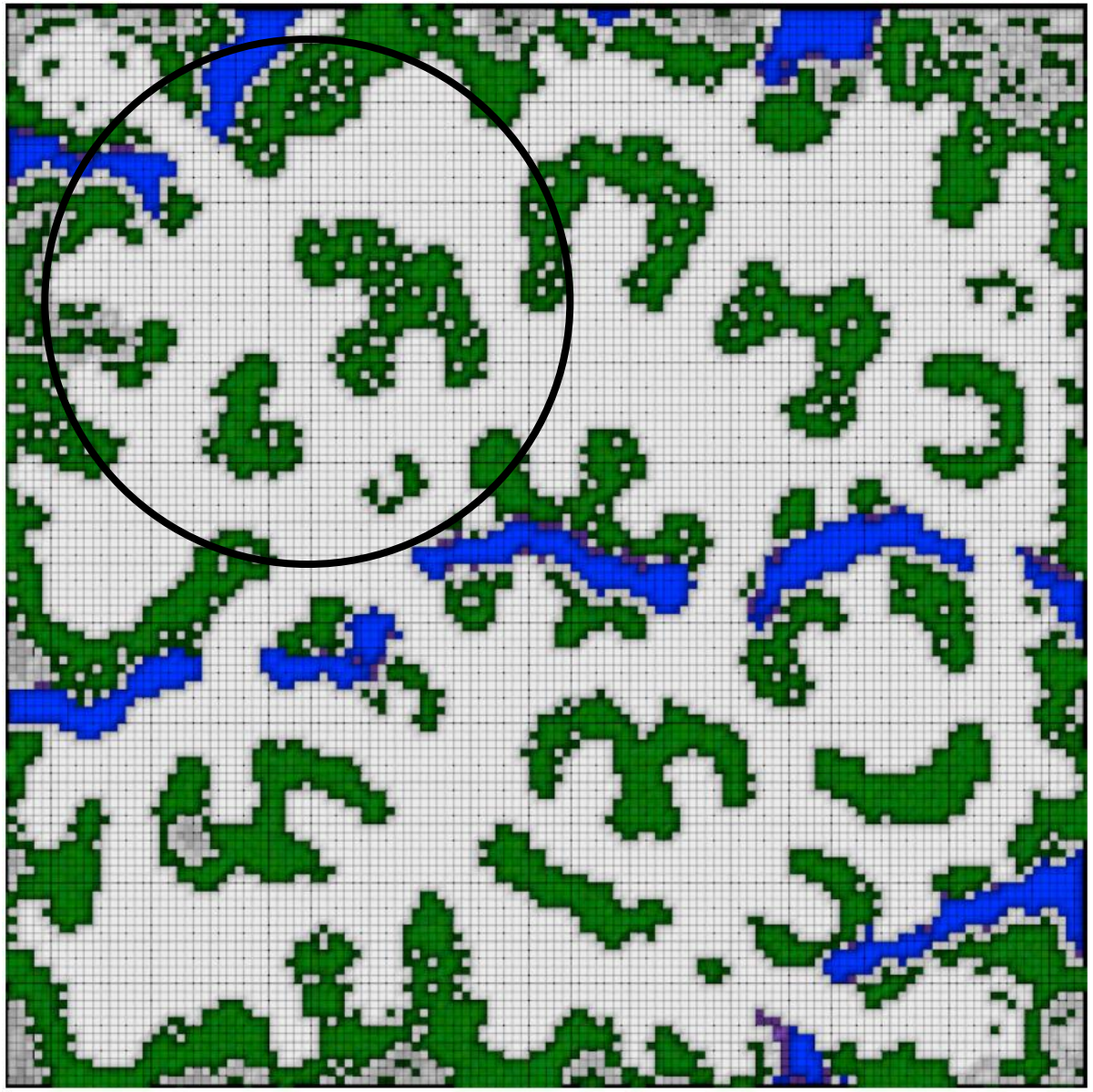
What are pros and cons of a path network representation of space?

Path network: pros

- Discretization of space is (can be) 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 (we will see this later)
- Continuous, non-grid movement in local area
- Switch between local and remote navigation
- Plays nice with “steering” behaviors (we will discuss these later)
- 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 destructible terrain
- Doesn’t fit well with map-generation features in games
- Fog-of-war in RTSs



Fog of war

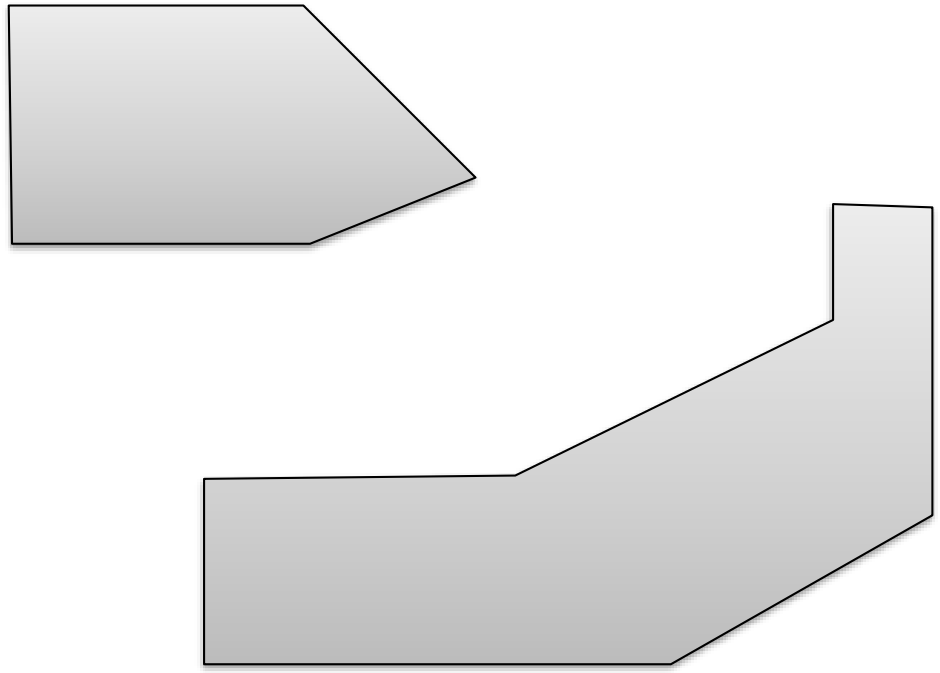
Path finding models

1. Tile-based graph – “grid navigation”
2. Path Networks / Points of Visibility NavGraph
3. **Expanded Geometry**
   - Discretization of space can be smaller
   - 2 tier nav: Continuous, non-grid movement in local area
   - Can work with auto map generation
   - Can plan nicely with “steering behaviors”
4. NavMesh

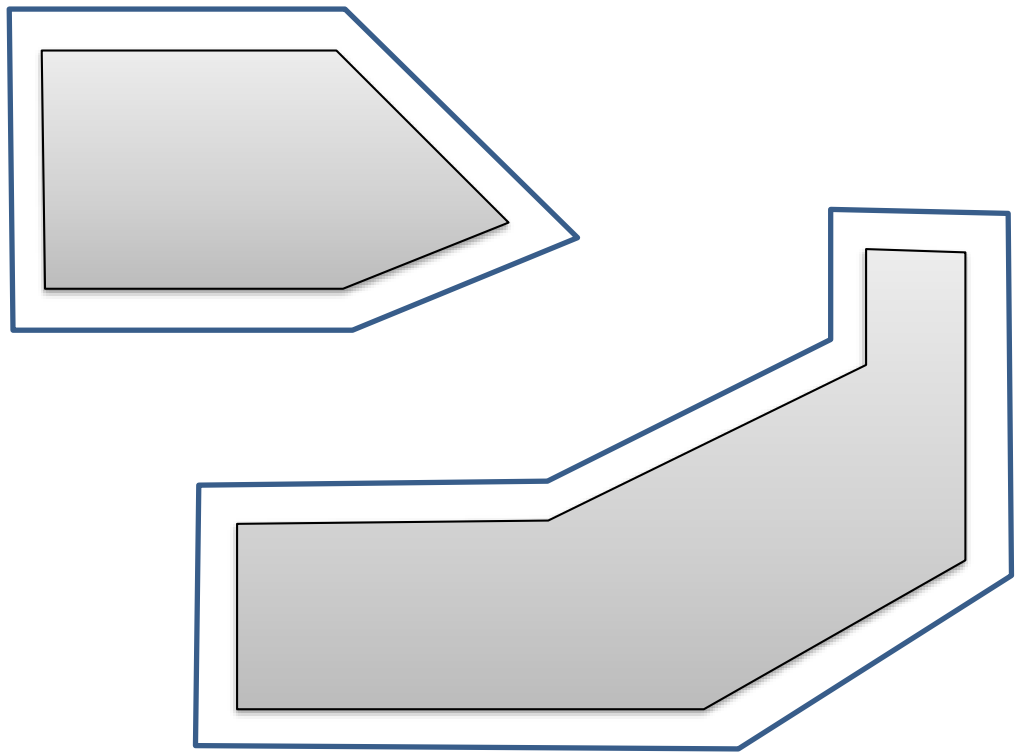
Model 3: Expanded Geometry

- Automatic, and no wall bumping.
- Also a two-tiered navigation system
  - Local, continuous
  - Remote
- 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

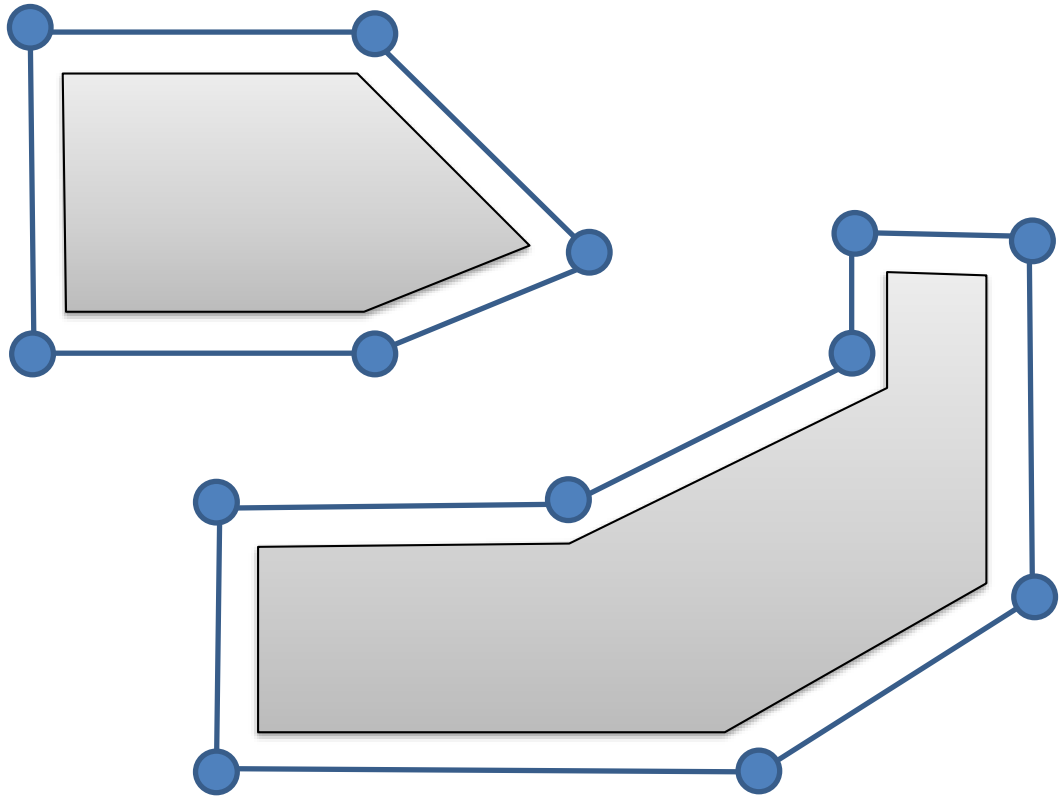
Simple Geometry



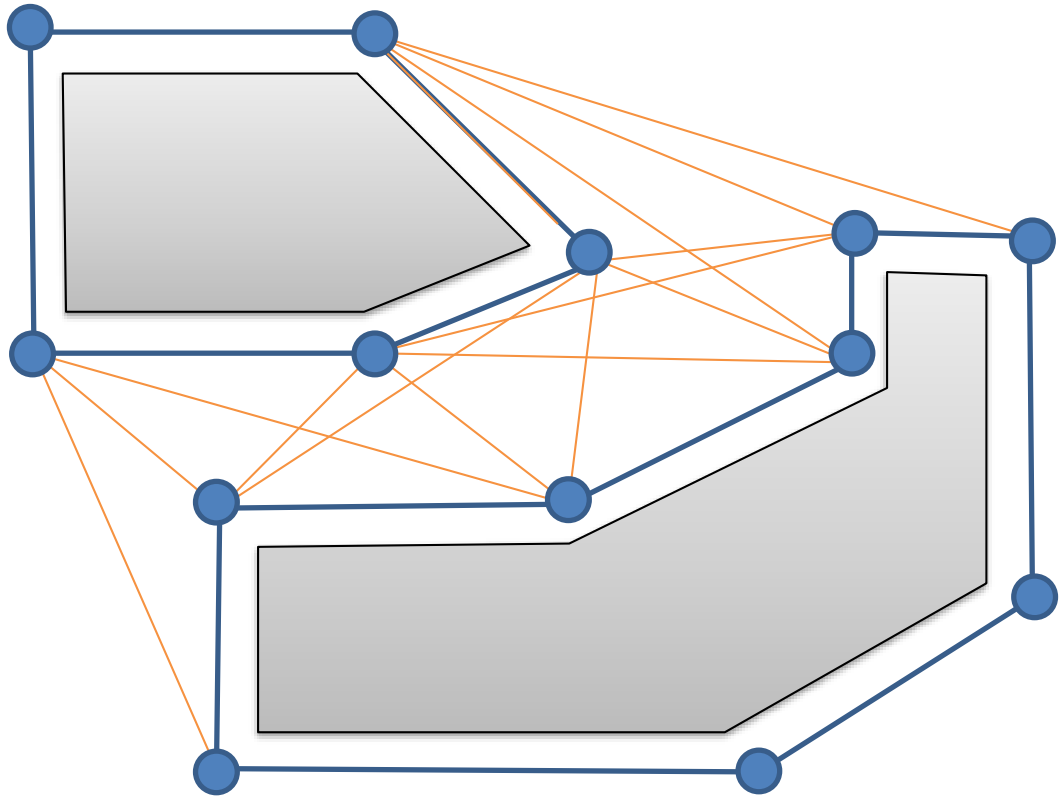
Expanded Geometry



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 destructible terrain (?)
- Fog-of-war in RTSs

Path finding models

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