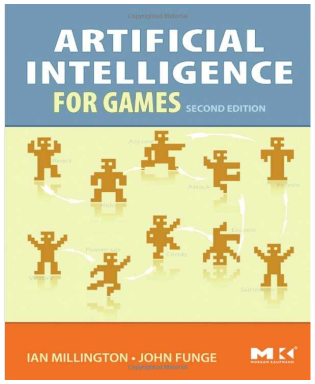
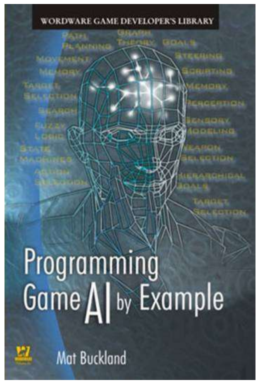



“the question of whether computers can think is like the question of whether submarines can swim” -- Dijkstra

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

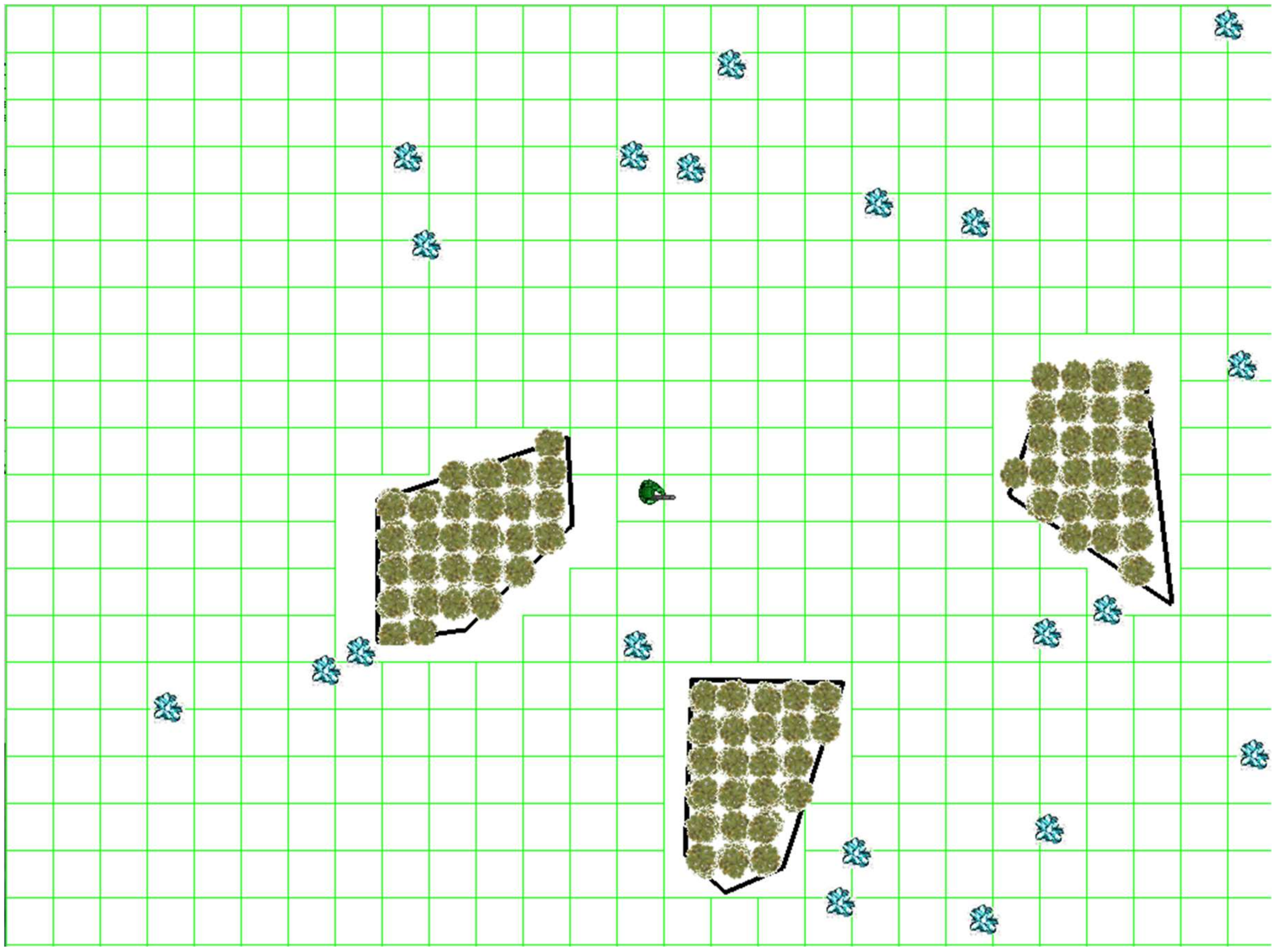
_____: A rule of thumb, simplification, or educated guess that reduces or limits the search for solutions in domains that are difficult and poorly understood.

Announcements

- HW1 due Sunday night, August 31 @ 11:55pm
- HW2 is much more challenging than HW1. Start early!
- Waitlist
- Game engine & HWs – piazza only; please no posting
- Office hours on shared calendar
  - https://calendar.google.com/calendar?cid=dGozaWc2ZGh1cTg0OG44aWQ3cGo5bDdlaG9AZ3JvdXAuY2FsZW5kYXIuZ29vZ2xlLmNvbQ
- Special lectures

Grid Generation Hints

- Verify no world line goes through grid lines (rayTraceWorld)
- Verify no obstacle point within grid cell? Grid corner in obstacle?
  - e.g. pointInside
- 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

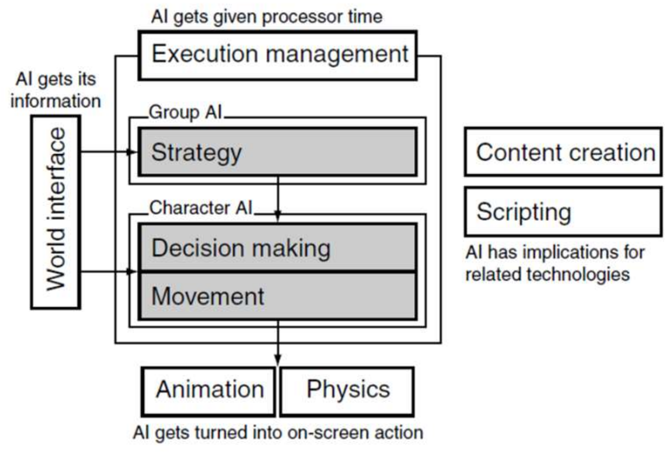
N-2: Common (game) “AI” Tricks?

- 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 graph 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

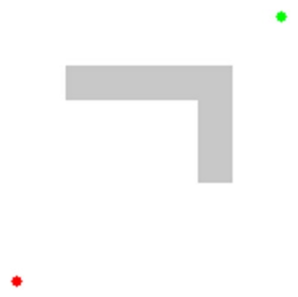
BRIEF RECAP OF LAST TIME

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*, D*-lite
- Hierarchical can help



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

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
- Height at center point of tile can be encoded in grid
  - Cost of navigation can be calculated based on gradient between neighbors

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

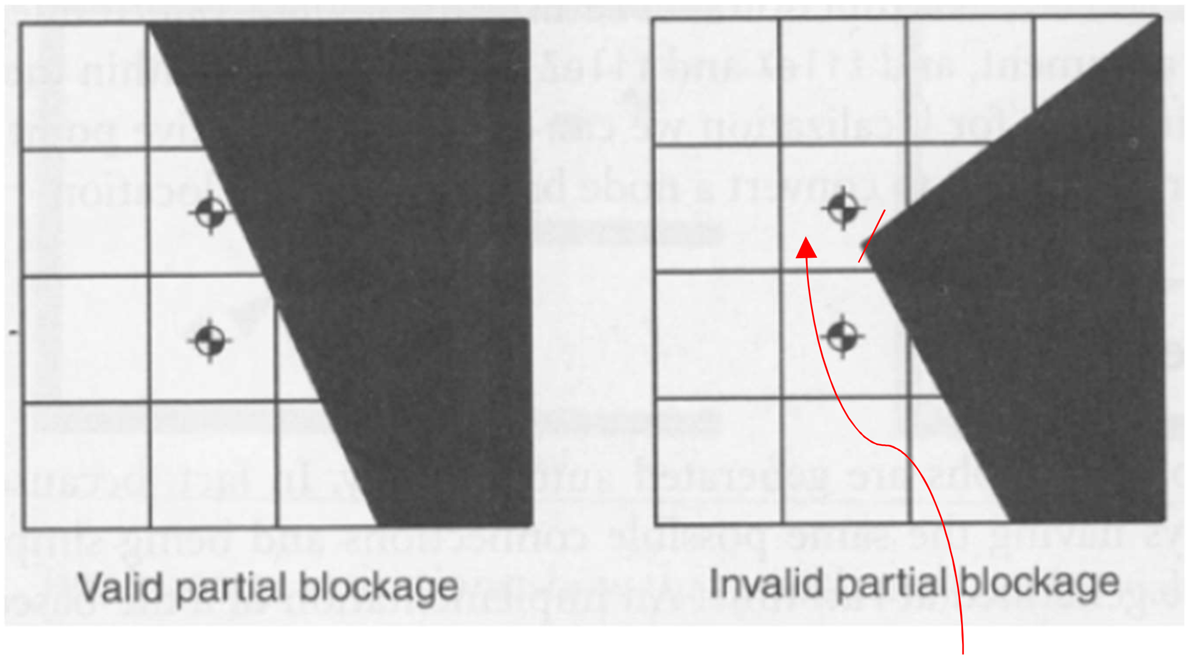
- Discretization “wastes” space
- Agent movement is jagged/awkward/blocky, though can be smoothed
- Some genres need continuous spaces
- Partial-blocking hurts validity
- Search must visit a lot of nodes (cells)
- Search spaces can quickly become huge
  - E.g. 100x100 map == 10k nodes and ~78k edges

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

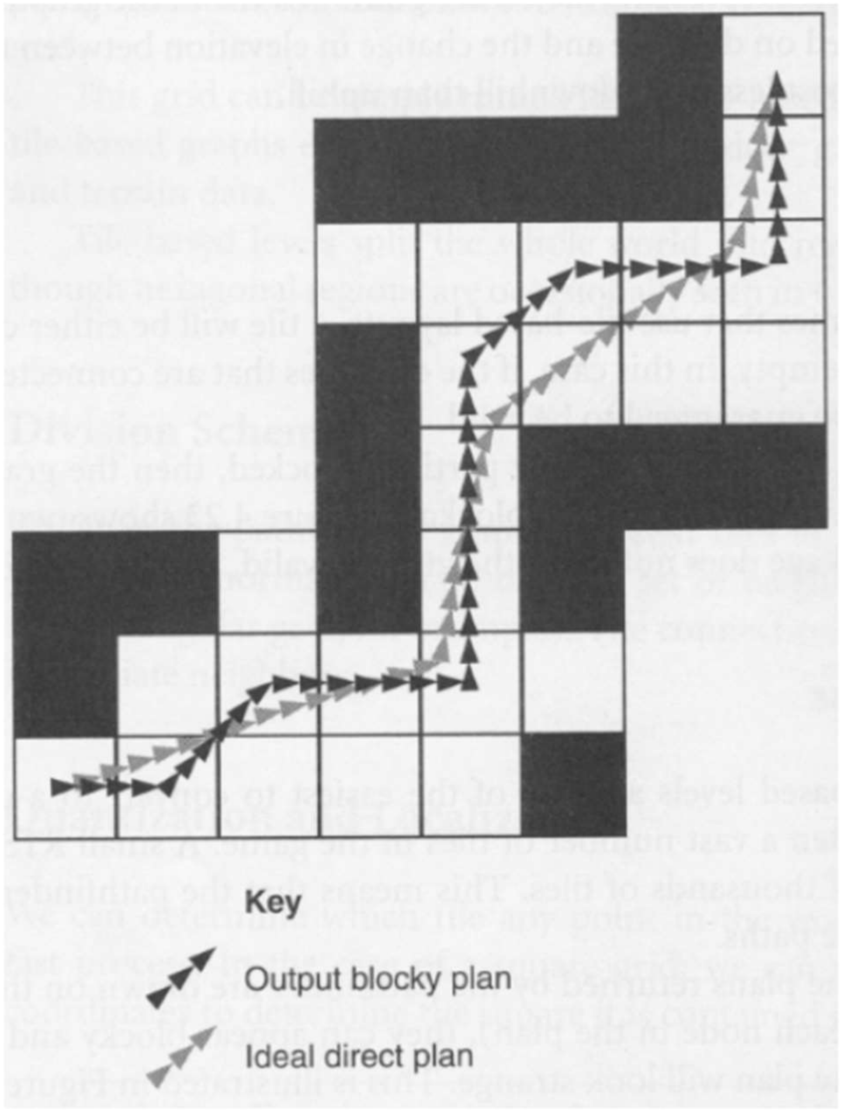
Validity

- Tile is typically completely blocked or completely empty
- If allow partial blockage, validity is risked



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

M&F 4.23



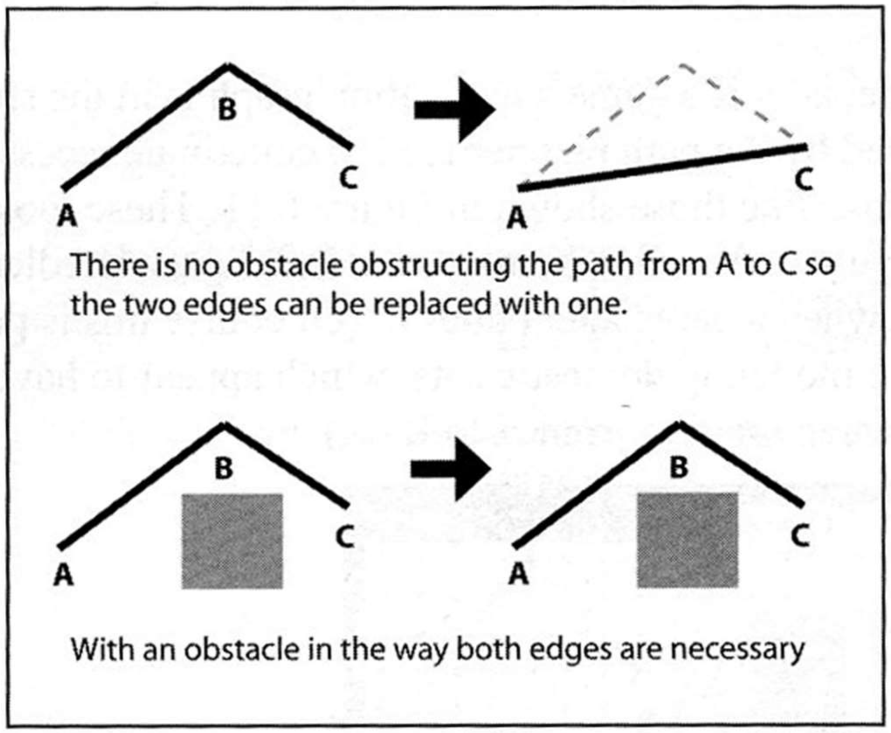
Fixing awkward agent movement:

- String pulling
- Simple stupid funnel algorithm
- Splines
- Hierarchical A*

M&F 4.24

Path Smoothing via “String pulling”

- 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

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.

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.

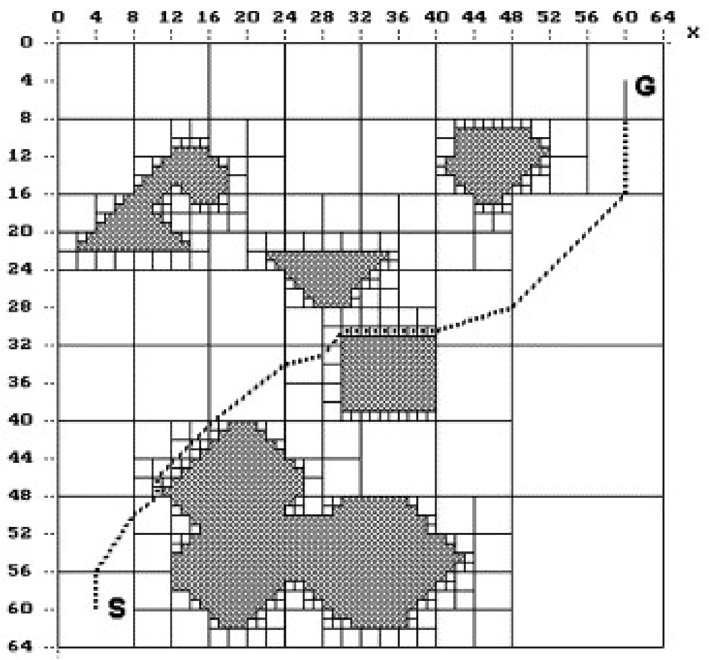
Long search times / Weird final paths

- Precomputing paths
  - Faster than computation on the fly
  - Especially with large maps or lots of agents
- Add extra heuristic to mark certain grid cells as more “costly” to step through.
  - Cells near obstacles
  - Cells that an agent can get “caught” on
  - Cells that an agent can get “trapped” in
- Path smoothing can help with weird final paths

See Also

- Binary space partitioning
- k-d tree
- Quad/Oct trees

- https://www.sciencedirect.com/science/article/pii/S0736584501000187
- I. L. Davis, “Warp speed: Path planning for star trek: Armada,” presented at the AAAI Spring Symposium (AIIDE), 2000, pp. 18–21.



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

2019-08-26;

See also: Buckland Ch 5 & 8,

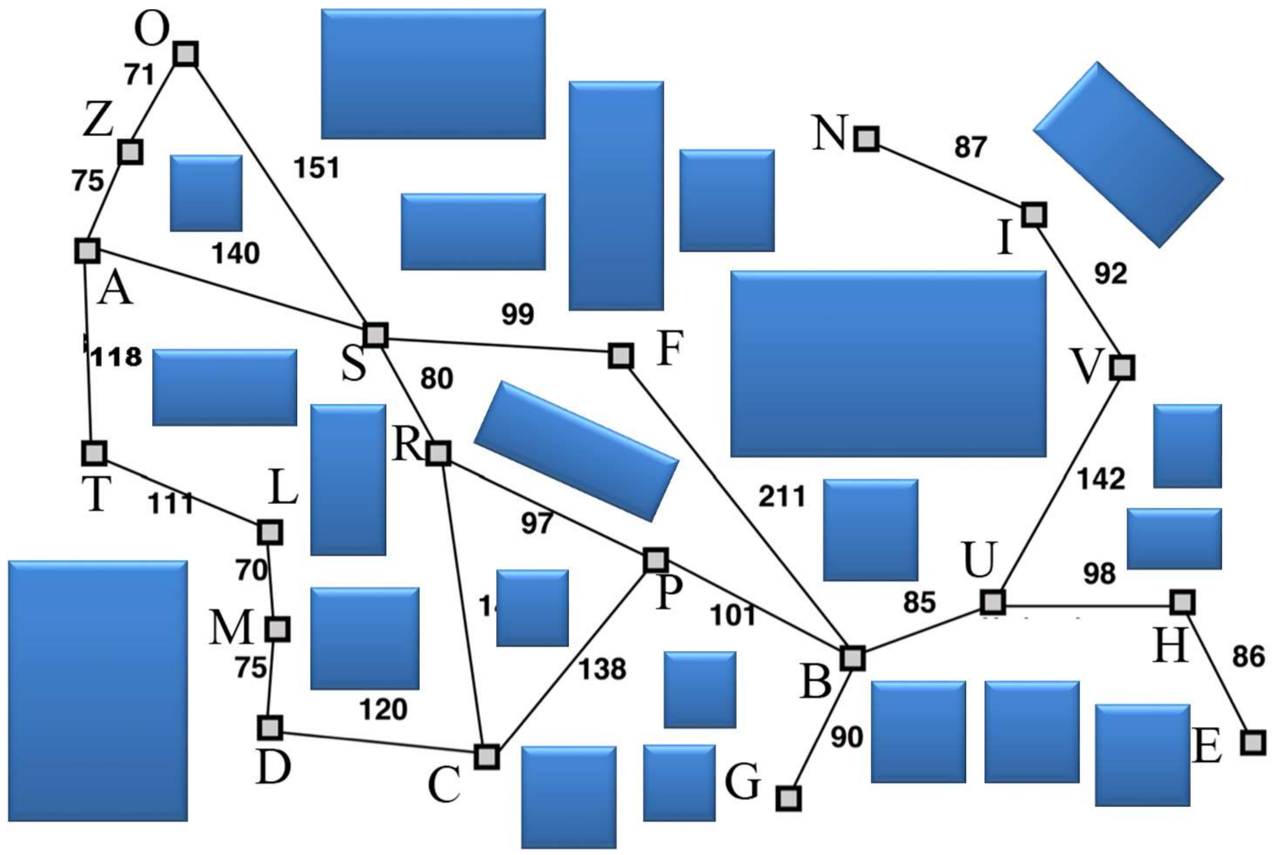
Millington & Funge Ch 4

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

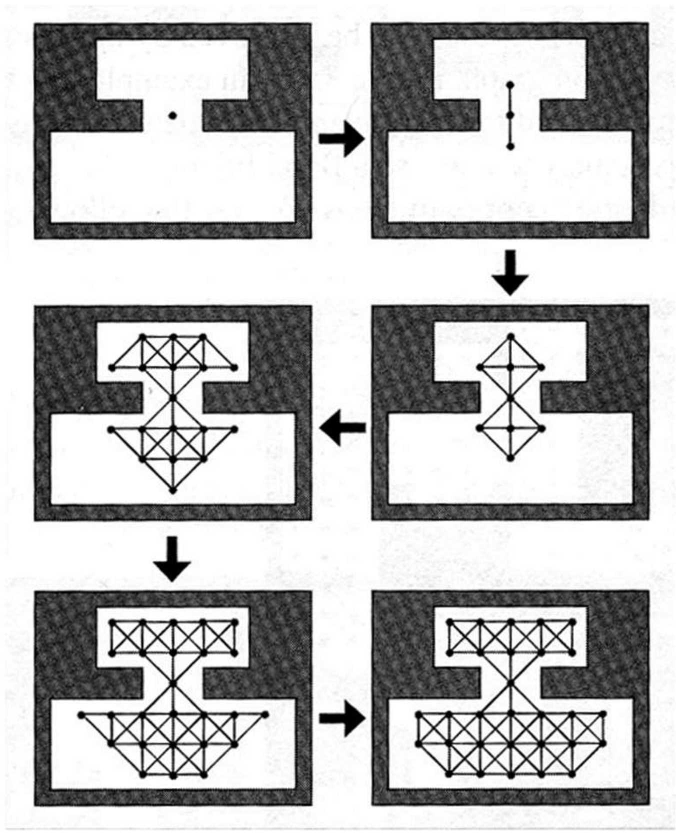
- POV: Points of visibility NavGraph (see B CH 8)
- Discretization of space into sparse network of nodes
- Two-tiered navigation system
  - Local, continuous
  - Remote
- Connects points *visible to each other* in all important areas of map
- Usually hand-tailored (can use flood-fill)


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



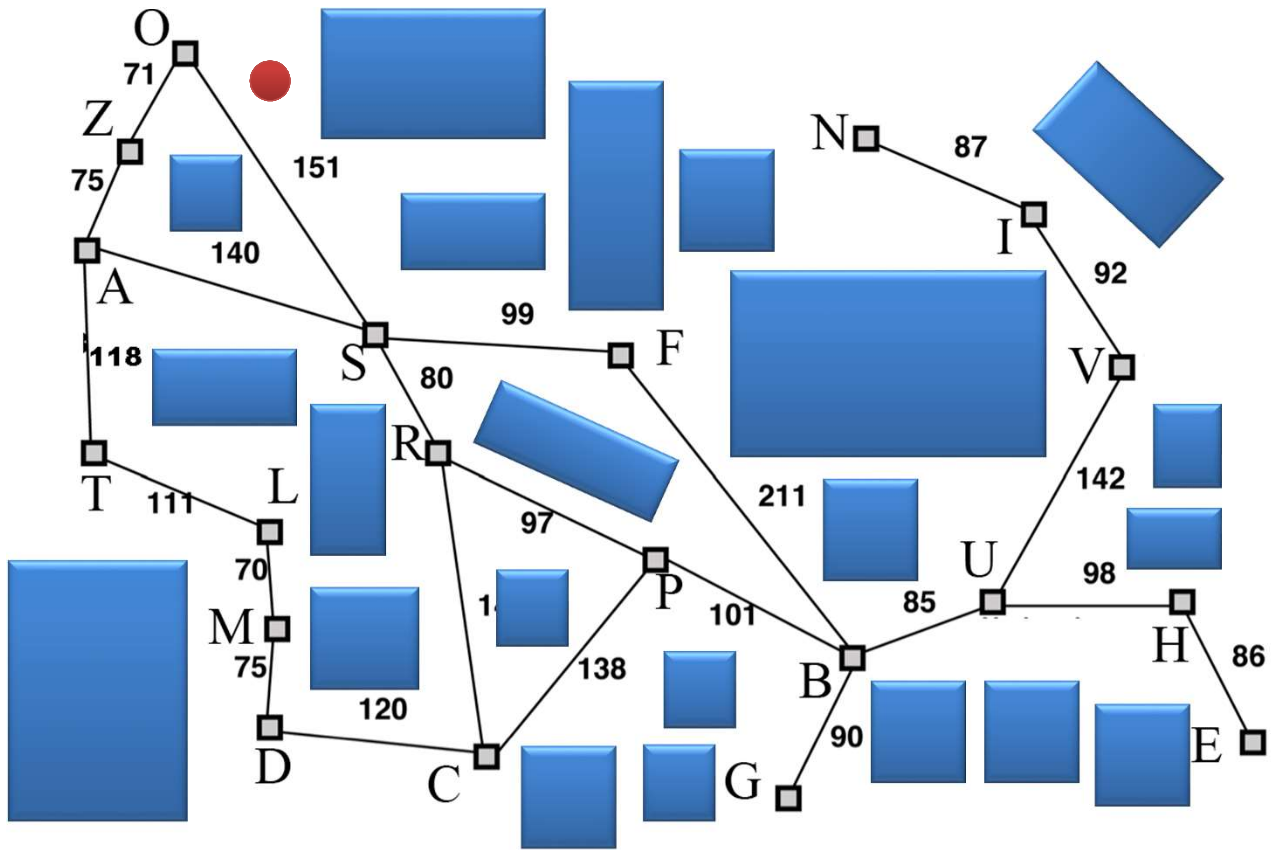
- Start with "seed"
- "grow" graph uniformly
- Designer can move, delete, or add nodes
- Ensure **all nodes and edges are at least as far from walls as agents bounding radius**

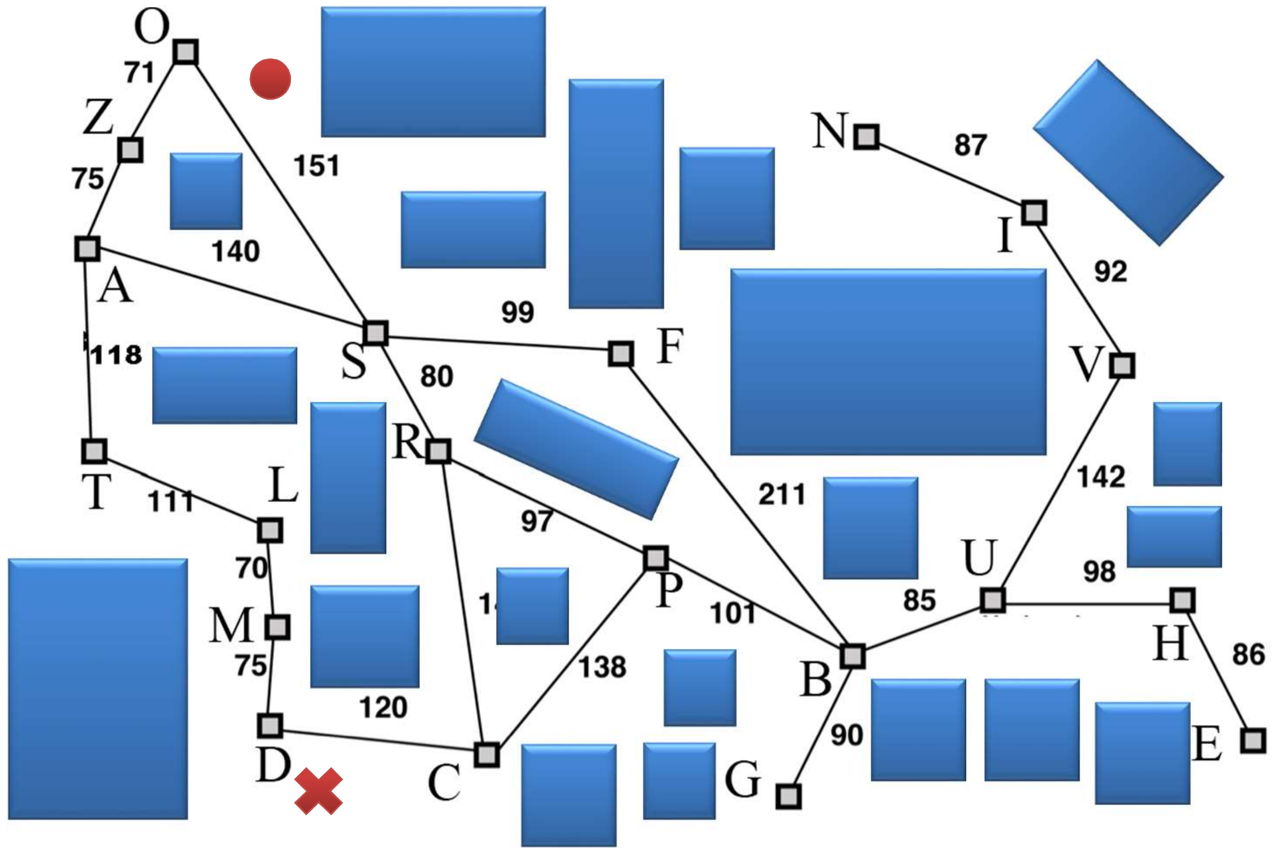
Buckland Figure 8.7

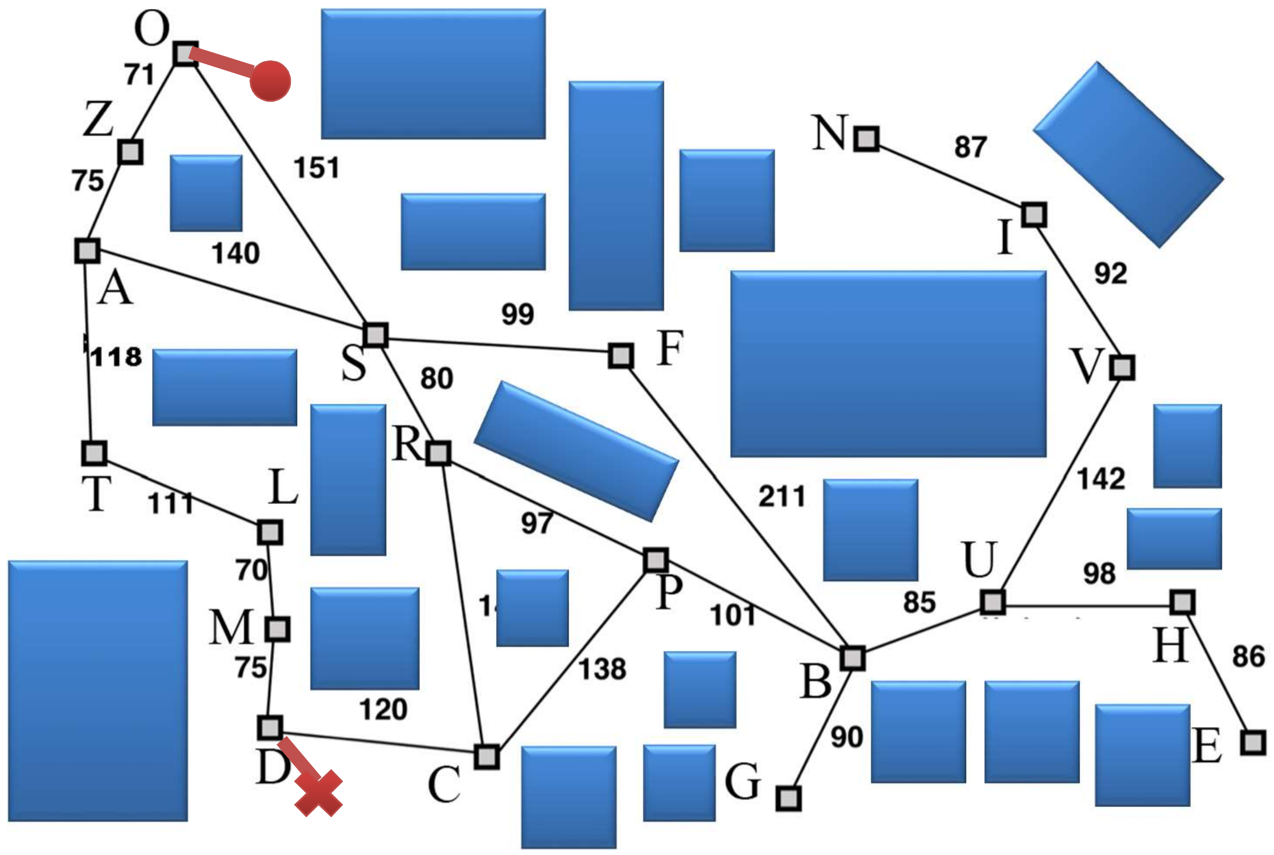
Using the path network

- 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

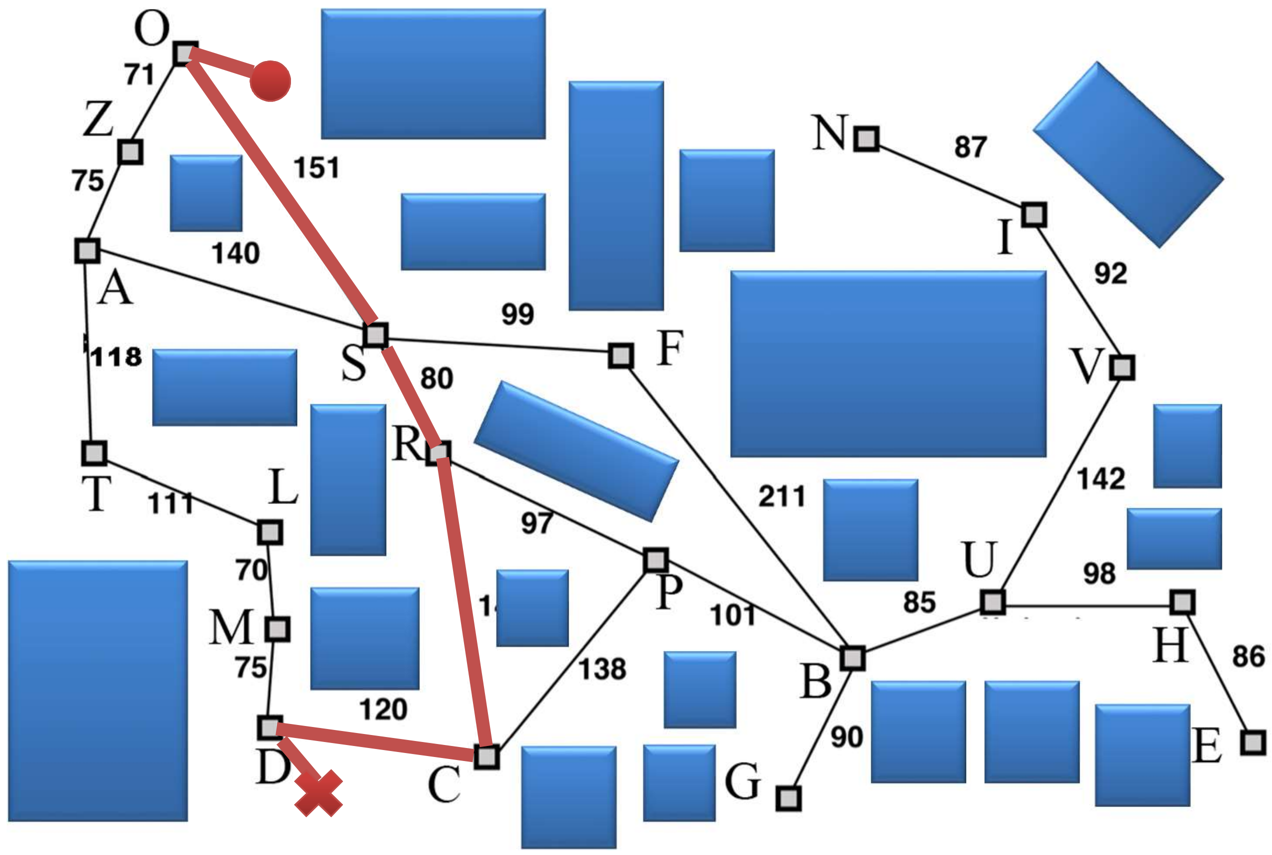
Problem: Unsightly paths.


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

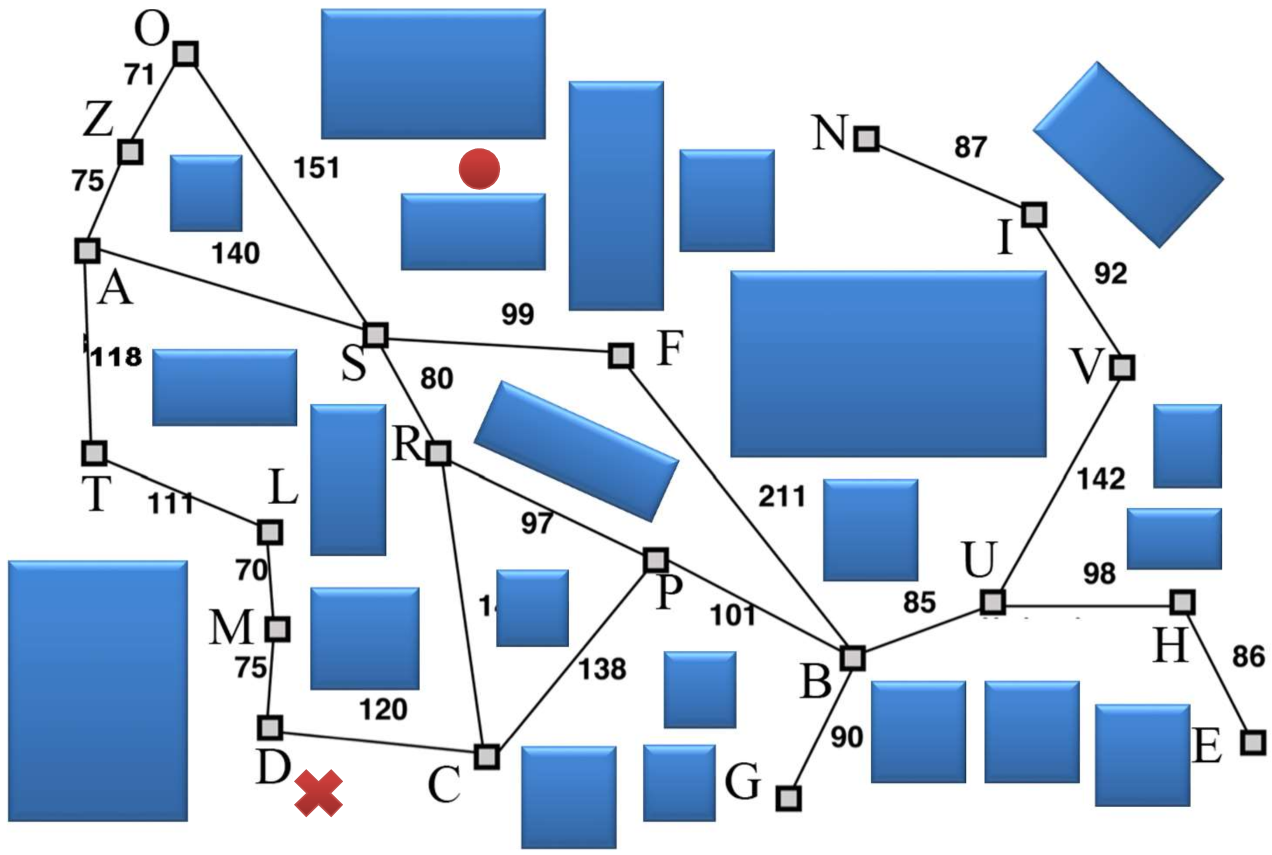

O
71
Z
75
151
140
A
99
118
S
F
80
R
T
111
L
97
70
P
M
101
75
138
120
D
C
G
B
90
N
87
I
92
V
142
211
U
98
85
H
86
E


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

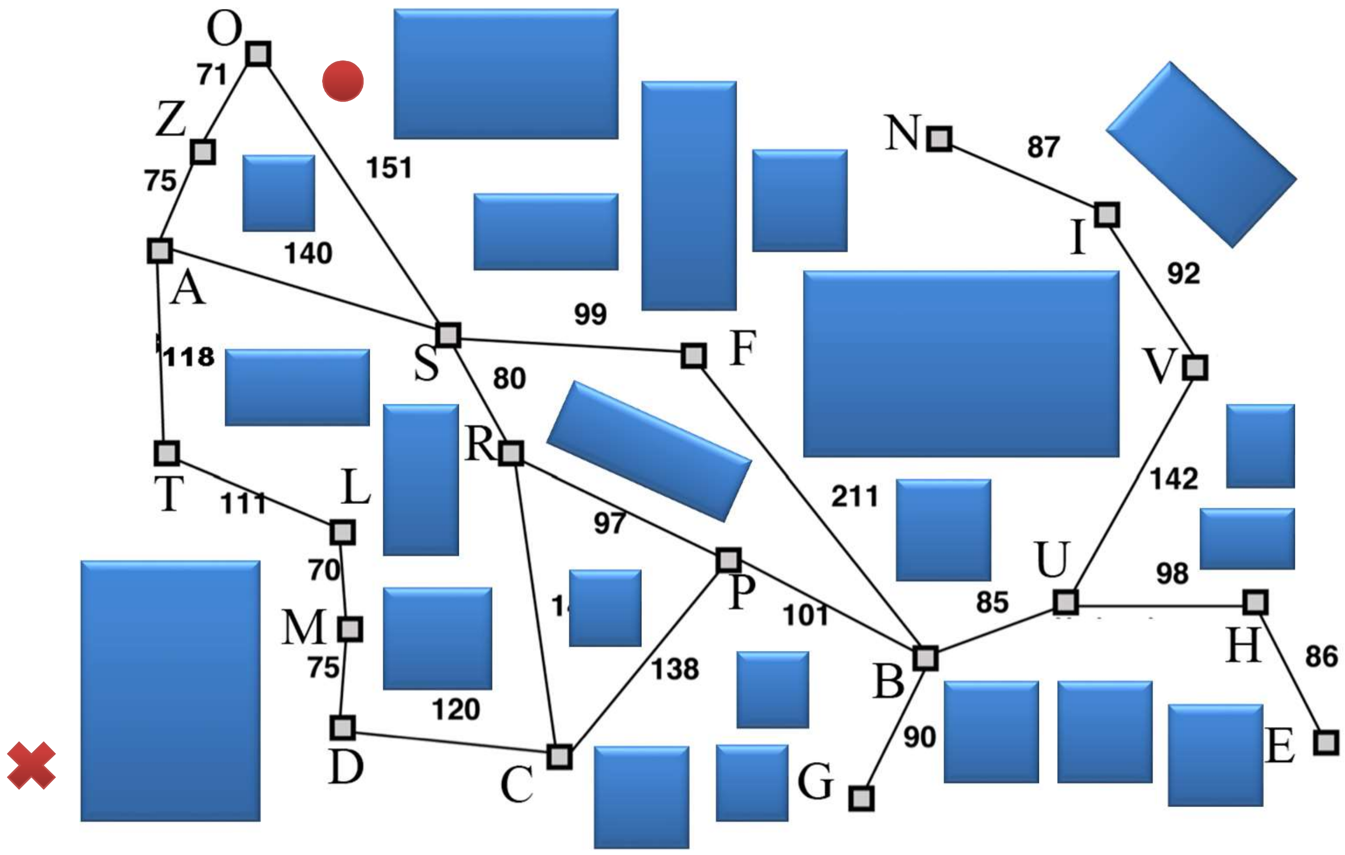
Unnecessary double-backs (unsightly)



Source not in sight of navigation point

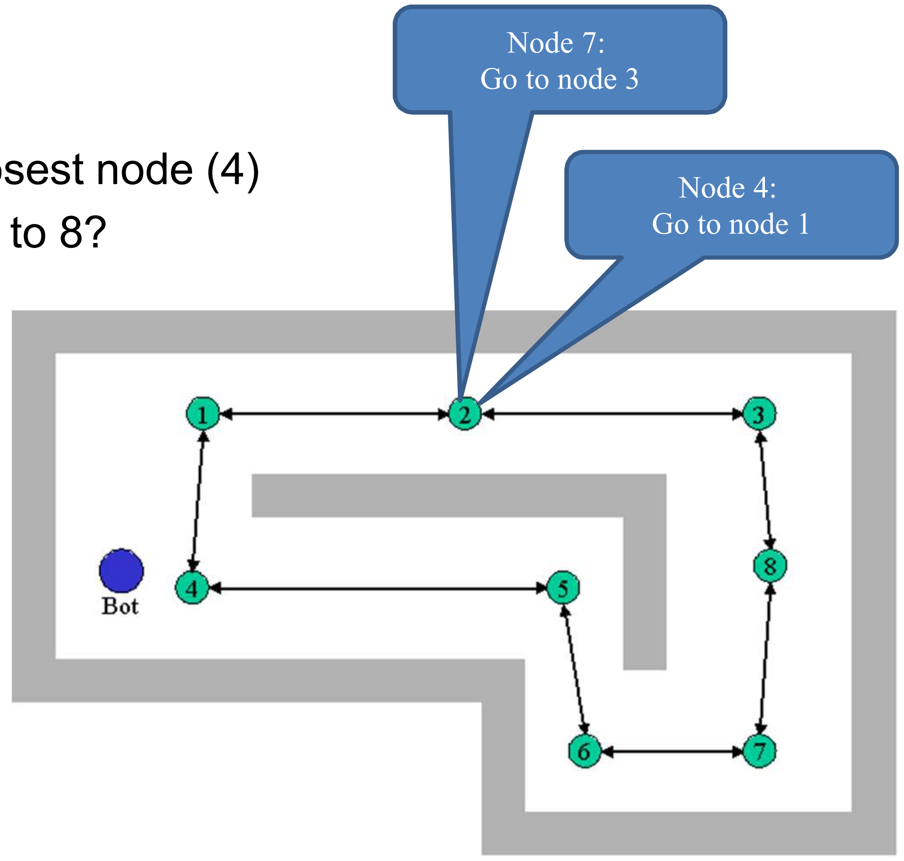


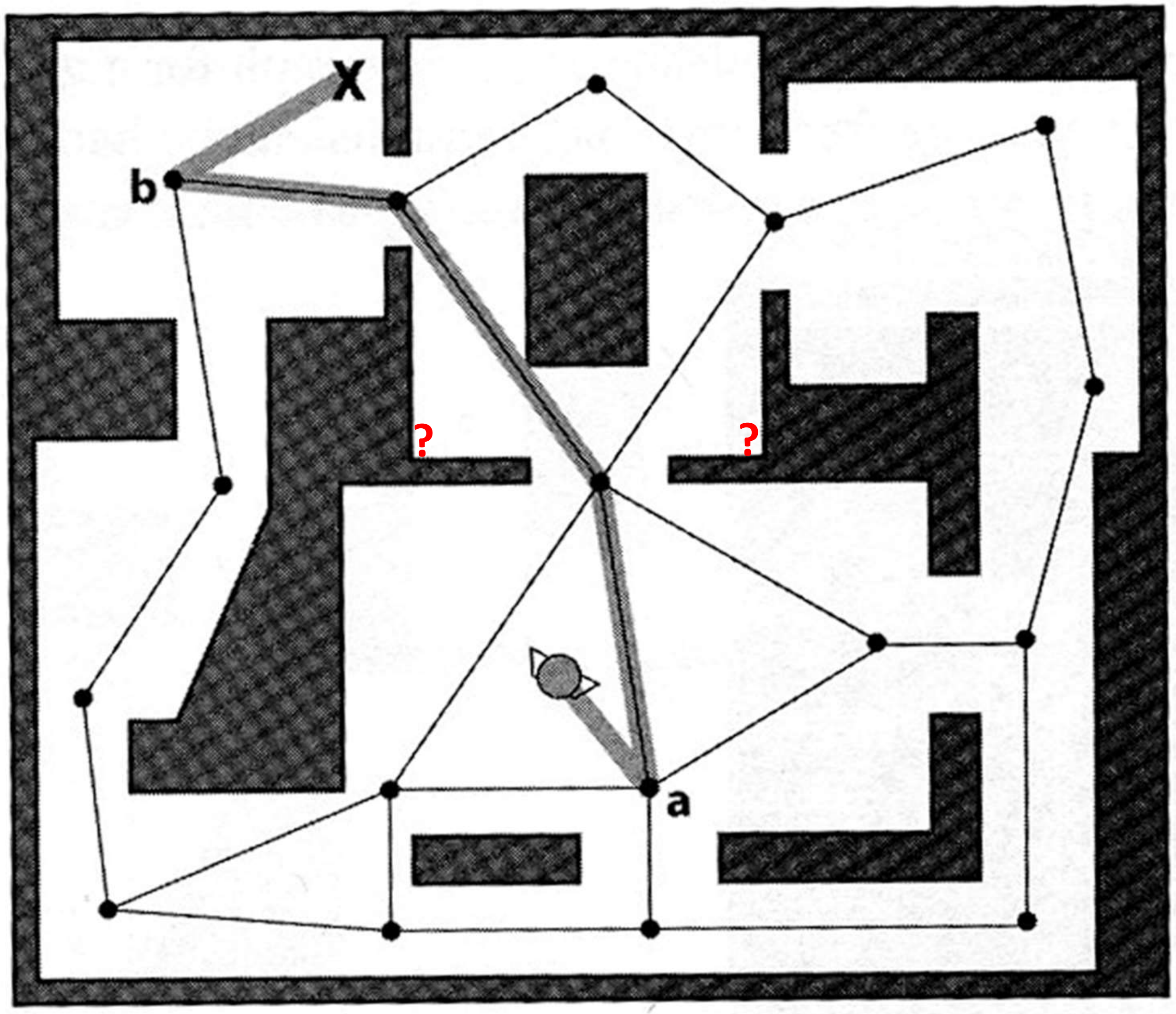
Destination not in sight of a navigation point

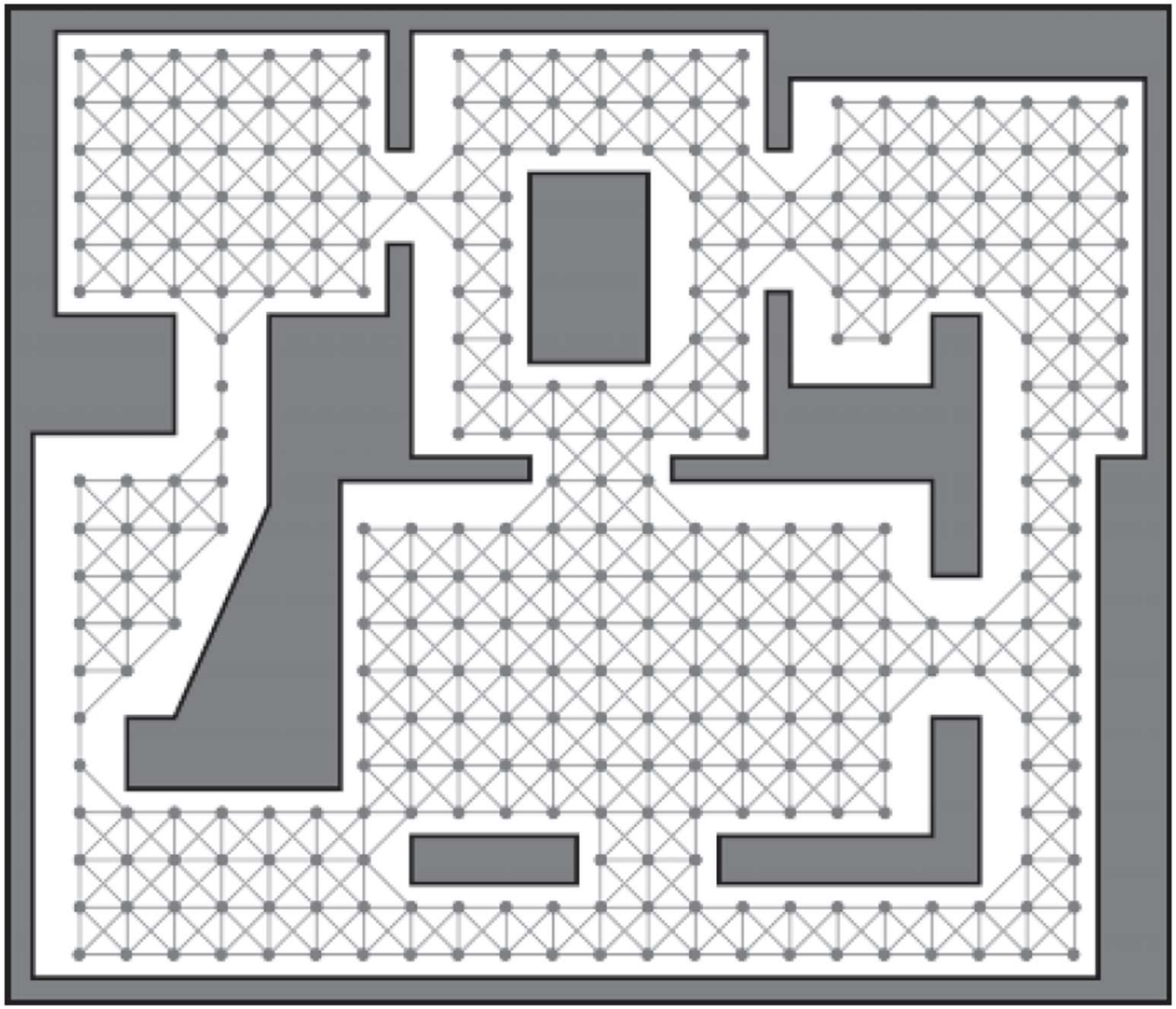


Using a path node network

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

Question

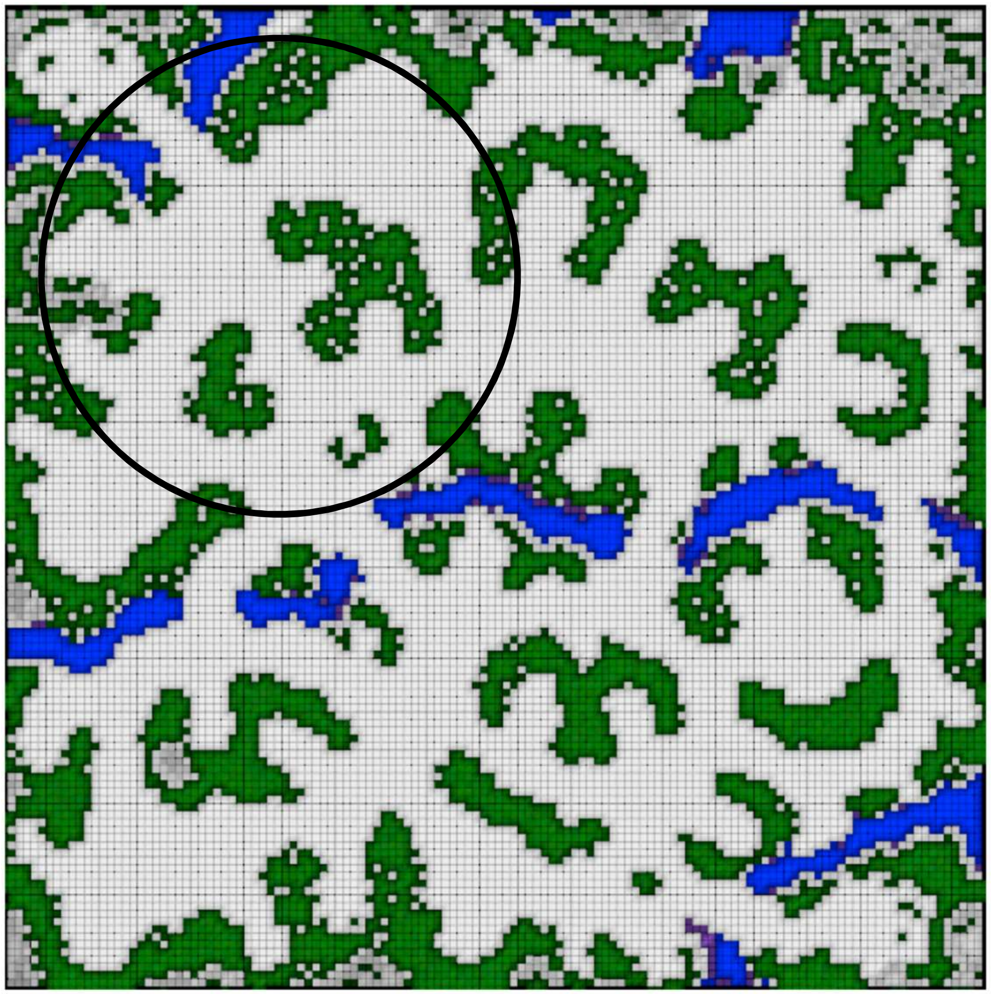
What are pros and cons of a path network representation of space?

Path network: pros

- Discretization of space is (can be – flood fill?) 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

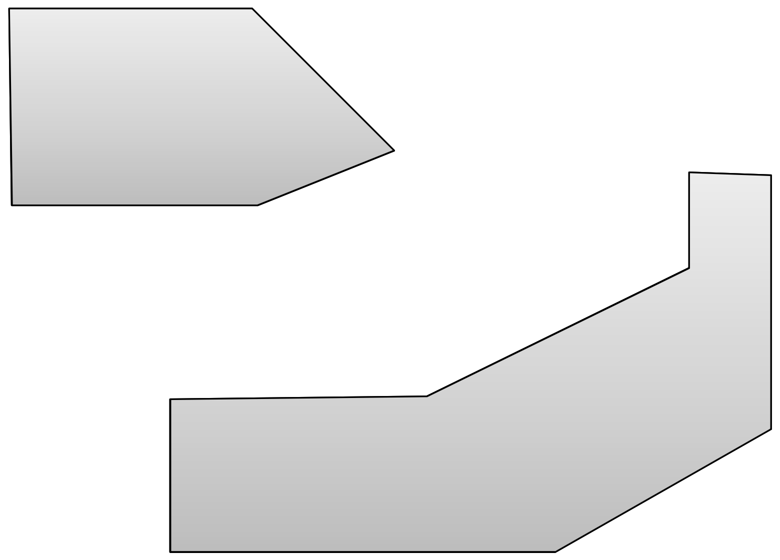
Path finding models

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

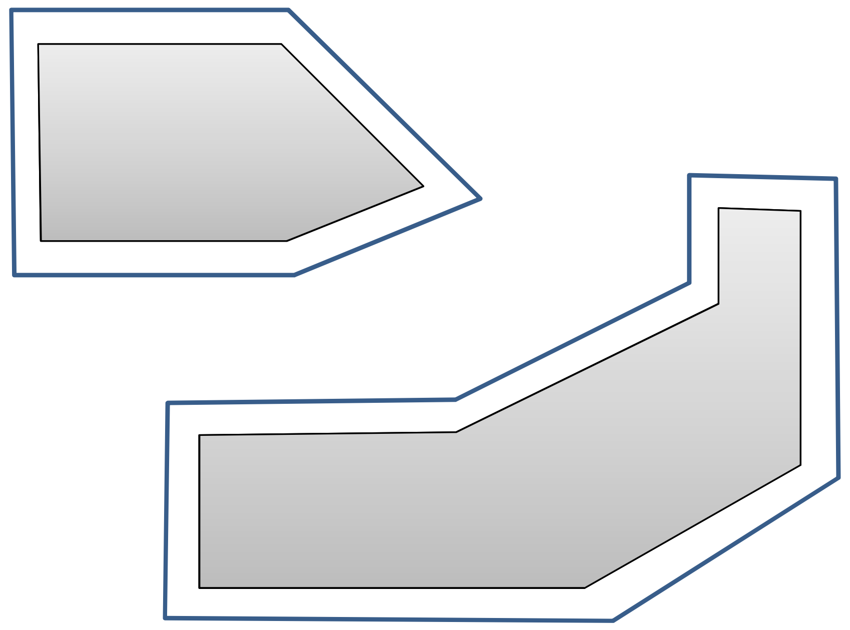
Model 3: Expanded Geometry

- 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

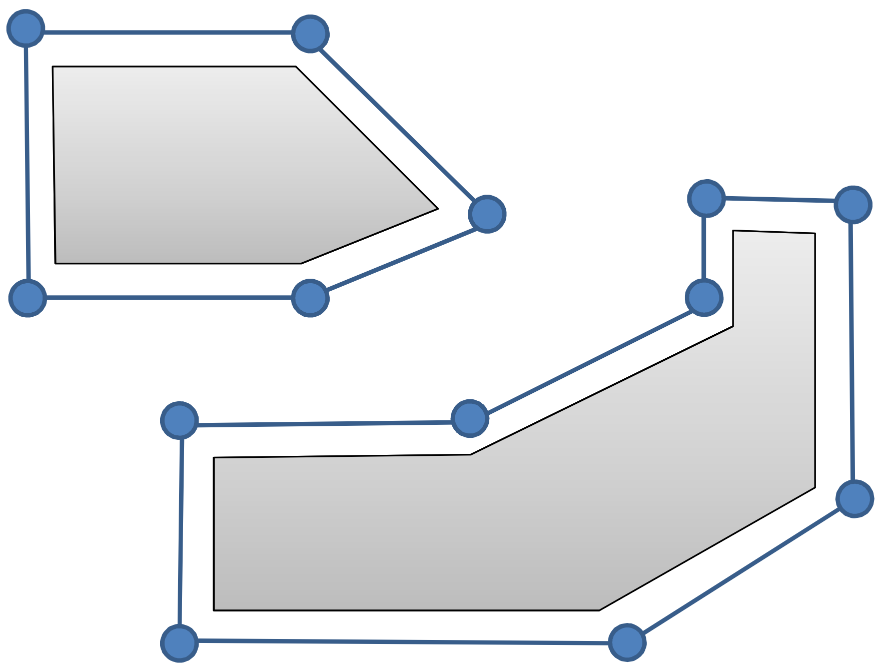
Simple Geometry



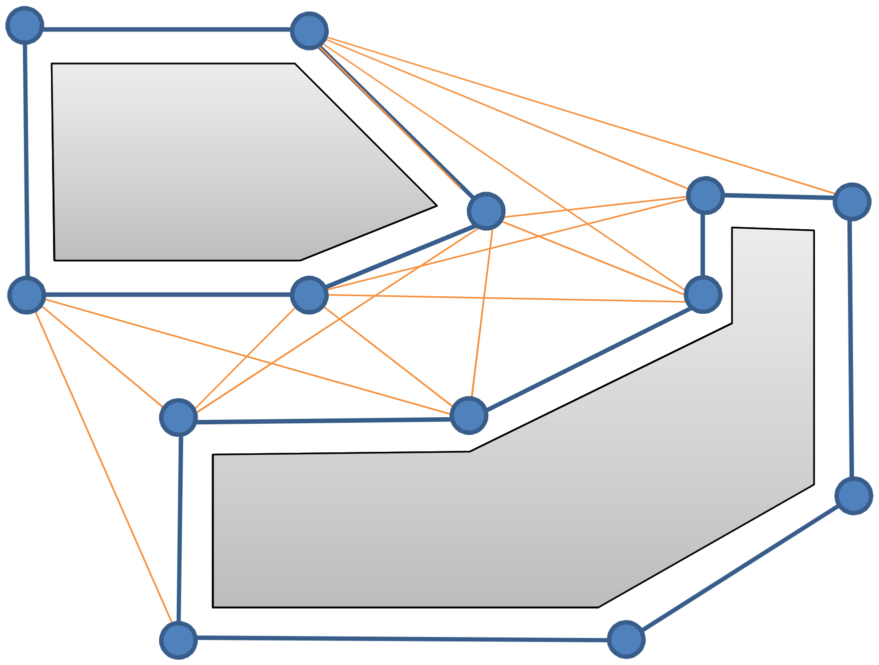
Expanded Geometry



Expanded Geometry

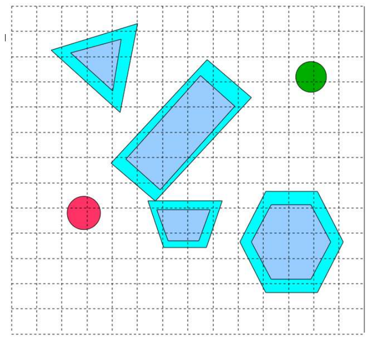


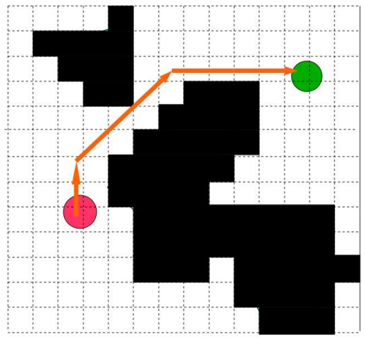
Finished POV Nav Graph



Expanded Geo: pros

- Discretization of space can be smaller
- Continuous, non-grid movement in local area
- Can work with auto map generation
  - Easier to bake into nav structures rather than recalculate
- Switch between local and remote navigation
- Can play nice with “steering” behaviors
- Applicable to all types of nav models!



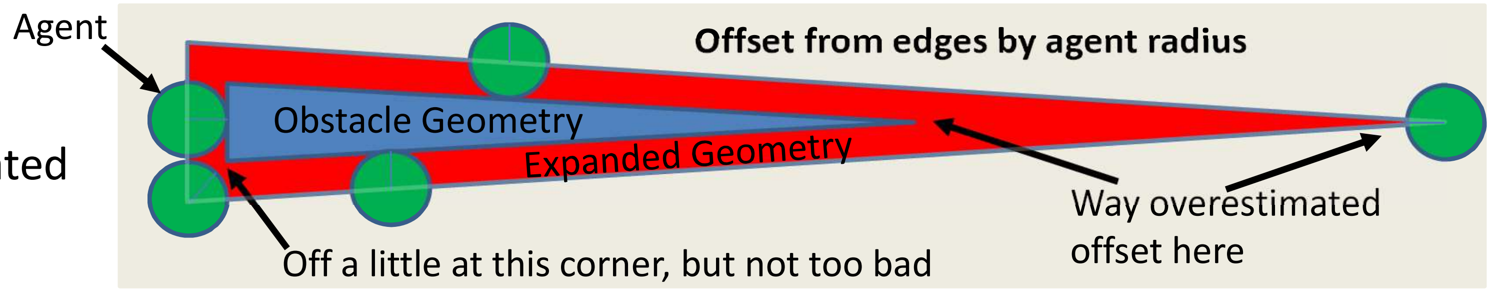


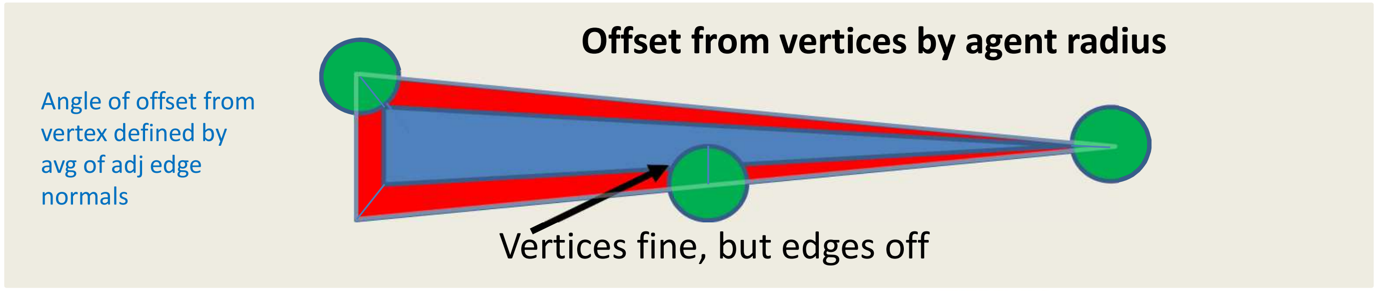
Expanded Geo: cons

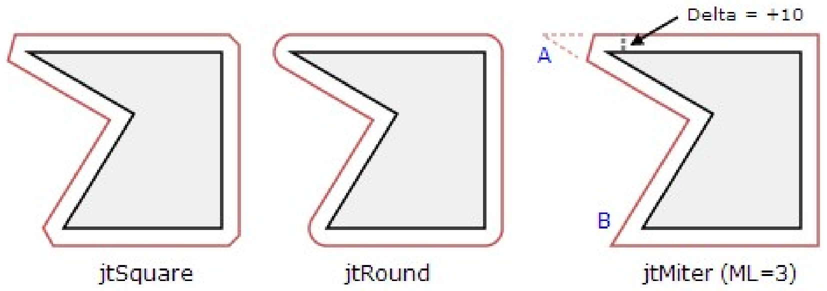
- Getting on and off the network can be even more awkward
- Dynamic pathing in destructible terrain (?)
- Fog-of-war in RTSs

Expanded Geometry: Corner"Gotchas"

- Expanding edges
  - can result in overestimated offsets
- Expanding vertices
  - can result in underestimated offsets
- Equidistant expansion
  - introduces non linear curvature (curved at corner offsets)
- Squaring off/selective mitering is compromise to avoid curves


Agent
Offset from edges by agent radius
Obstacle Geometry
Expanded Geometry
Way overestimated offset here
Off a little at this corner, but not too bad


Offset from vertices by agent radius
Angle of offset from vertex defined by avg of adj edge normals
Vertices fine, but edges off



The angle at vertex 'A' is more acute than that at 'B' and, since a mitered offset would exceed the specified limit (3 x delta), its offsetting is 'squared'.

Credit: Jeffrey Wilson

See also

- 12:00 https://www.gdcvault.com/play/1024912/Beyond-Killzone-Creating-New-AI
  - Navmesh, waypoints, string pulling, a*, Bezier path smoothing, steering behaviors, polygon vs point paths
  - http://digestingduck.blogspot.com/2010/03/simple-stupid-funnel-algorithm.html
    - https://www.gamedev.net/forums/topic/669843-the-simple-funnel-algorithm-pre-visited/
  - http://jeffe.cs.illinois.edu/teaching/comptop/2009/notes/shortest-homotopic-paths.pdf

Path finding models

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