AI for games

Lecture 3

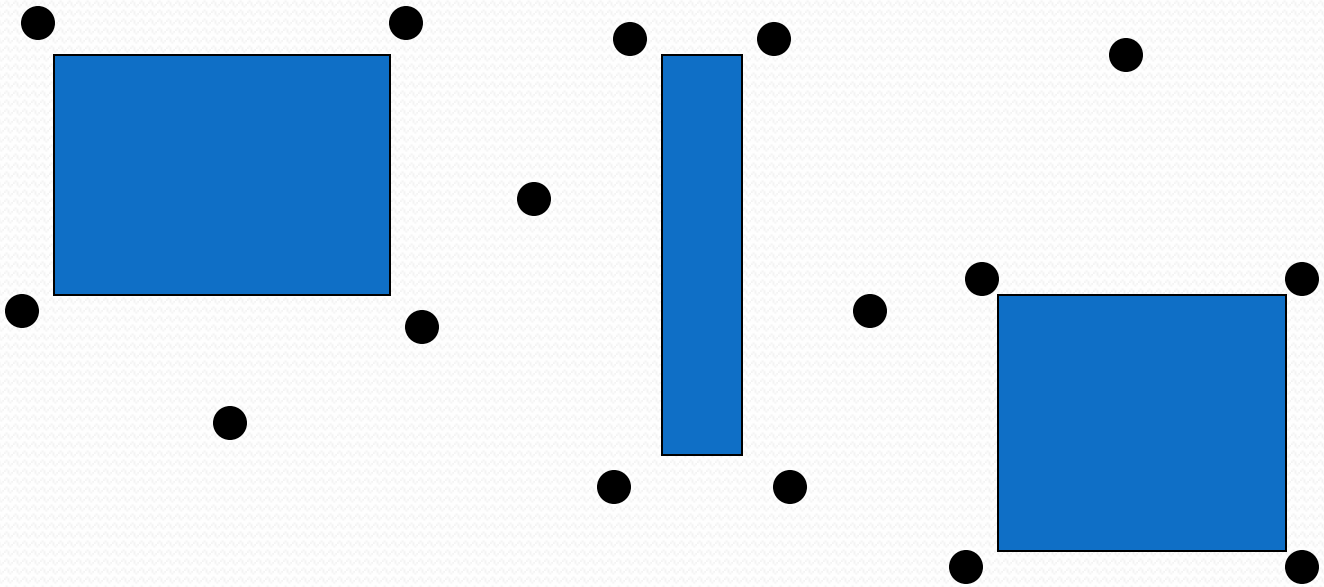
Nodes (waypoints)

Organiser

- Nodes
- Node graphs
- Partitioning
  - Rectangular grid
  - Points of visibility
  - Expanded geometry
  - Partitioning
  - Navmesh

Nodes

- A “node” is a point on the map that the character can plan a path to.



Nodes

- This does not mean that the character can't go anywhere else.
  - It just means that the AI will need to find the closest (possibly) node to the start point and destination and plan from there.
- This lecture looks at how to find and place these nodes.

Node graphs

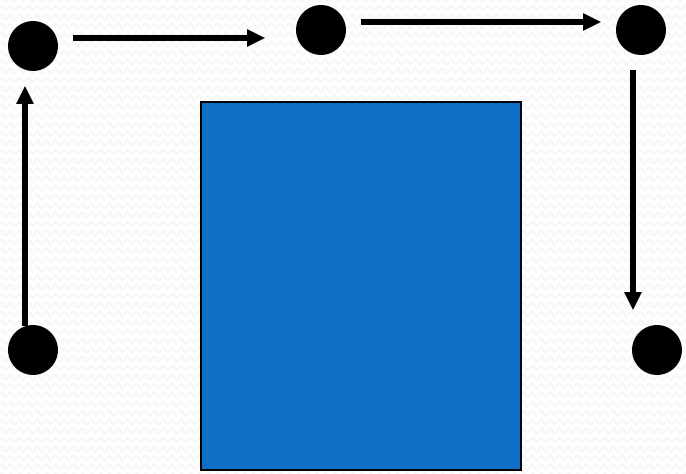
- The reason for having these nodes, is that **before the game runs**, we can calculate whether any node can see any other node, and the distance between them.

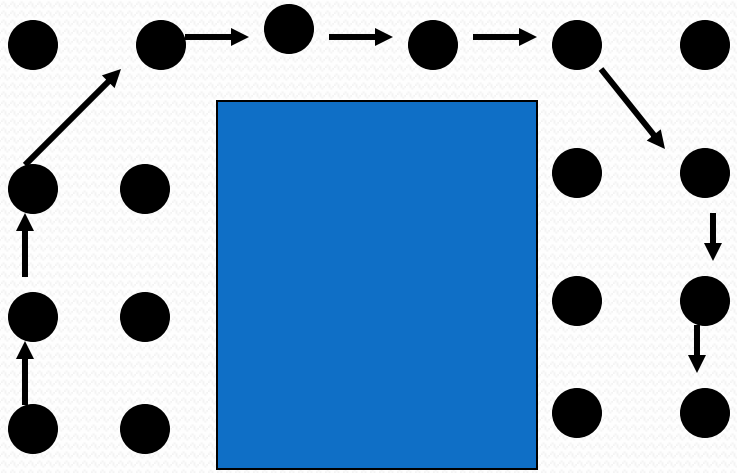
Visibility table

Node	A	B	C	D	E	F	G
A	-	12	20	-	-	-	9
B	12	-	-	46	-	-	-
C	20	-	-	12	-	-	-
D	-	46	12	-	9	72	-
E	-	-	-	9	-	12	-
F	-	-	-	72	12	-	12
G	9	-	-	-	-	12	-

Node graphs

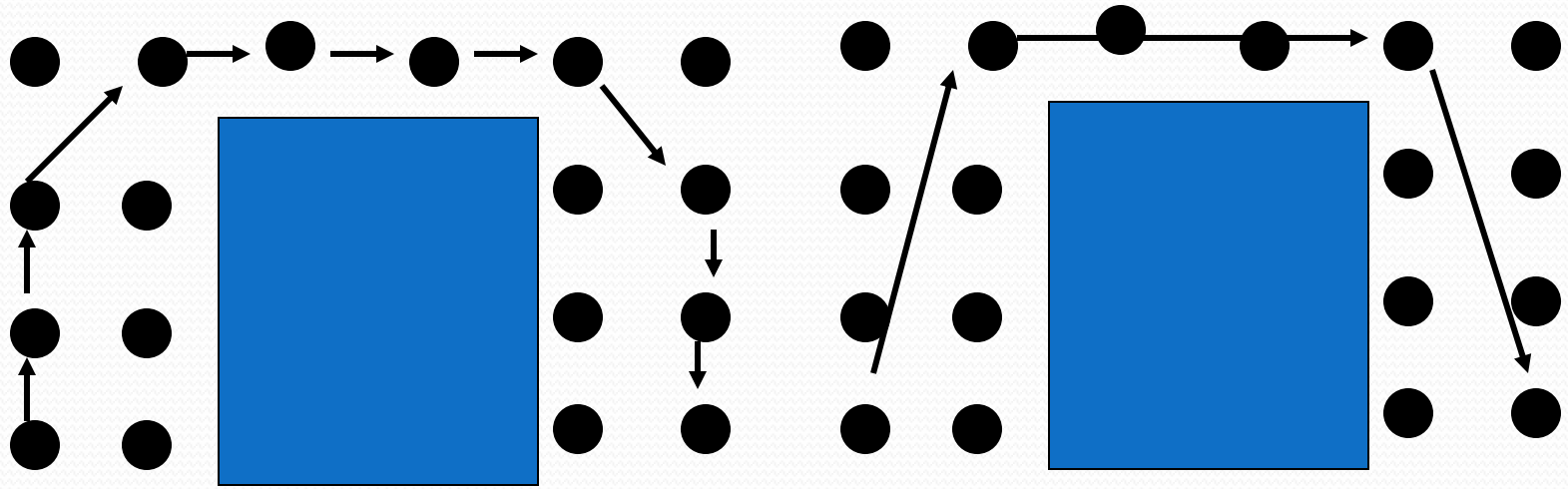
- Trade-off
  - Too many points makes the path-finding slow.
  - Too few points makes the paths found inefficient.





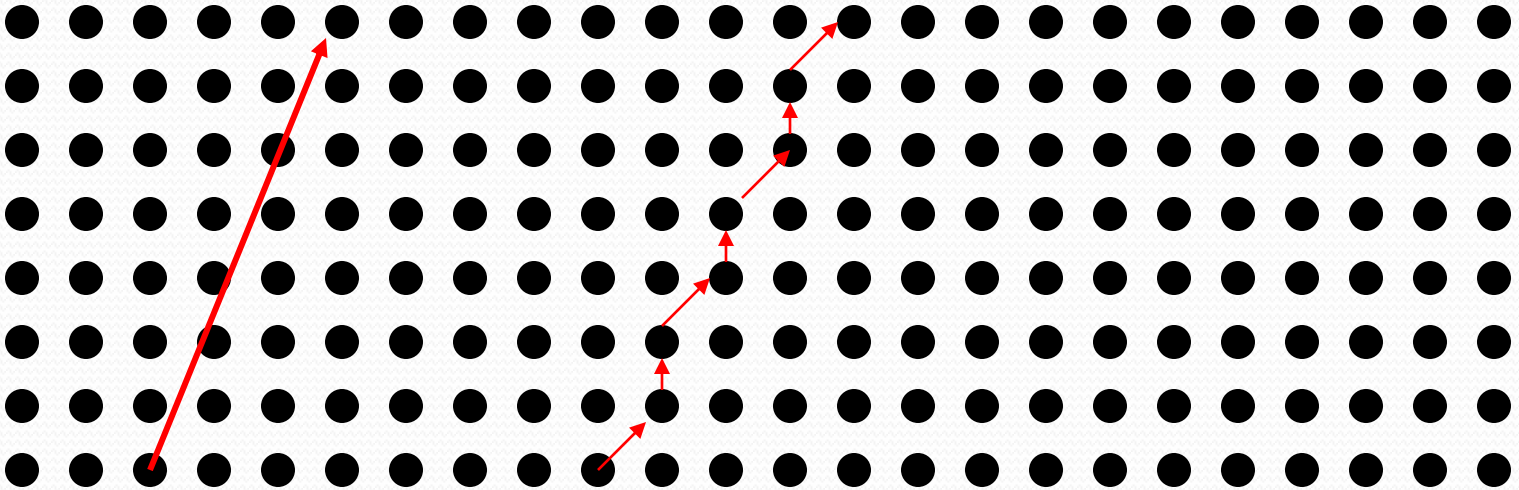
Node graphs

- Second trade-off
  - Can make the path more efficient by showing all visibilities.



Node graphs

- This also makes paths smoother



Node graphs

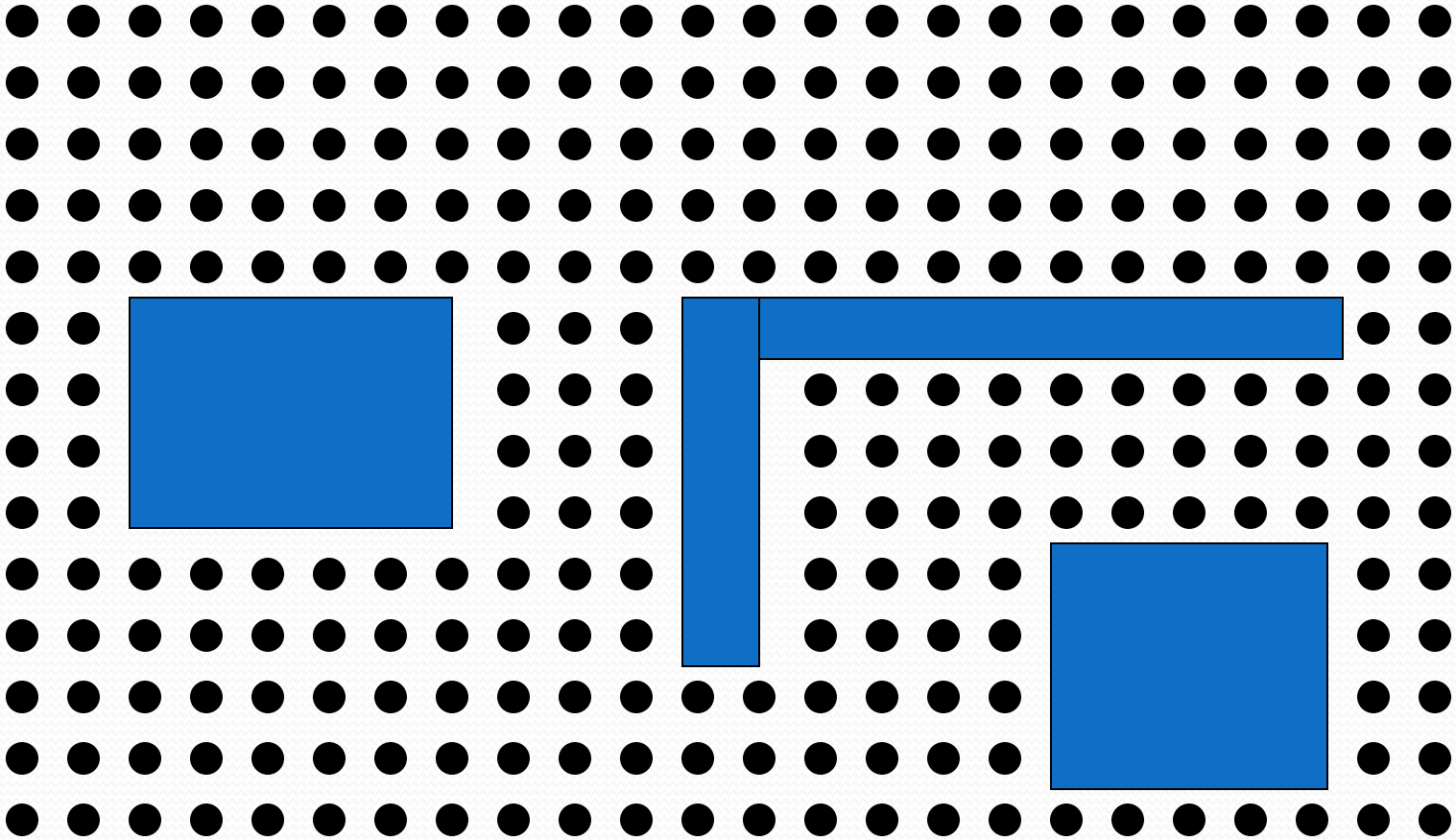
- But!
  - This makes the visibility table less sparse.
  - Greatly increases complexity of the pathfinding.
- On previous slide, each node has to check paths on 6 other nodes.
  - Or 144.

Node graphs

- This can largely be avoided by placing the nodes carefully.
- And by using some sort of path smoothing after generating the route.

- So how do we place the nodes?

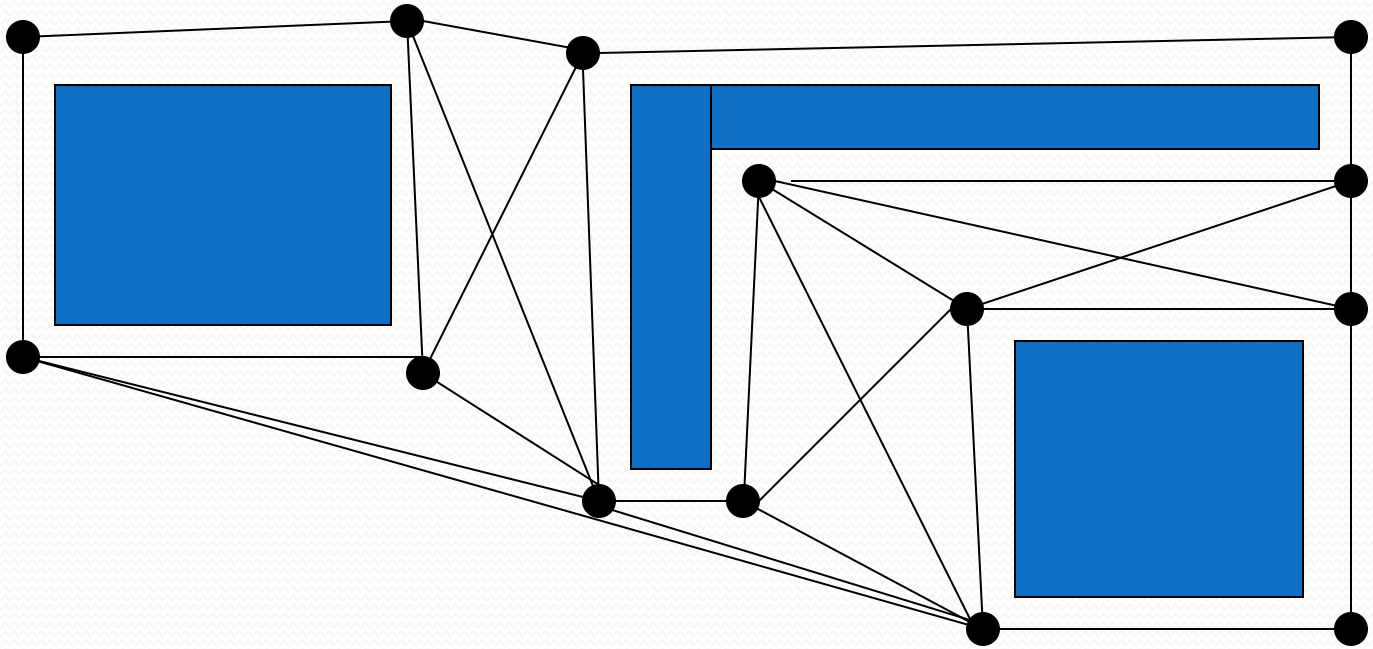
Rectangular grid



Rectangular grid

- Does the job, but has lots of nodes.
- Fine for cramped conditions, but gives you lots of surplus nodes in any open areas.
- For some games, you can use hexagons, not squares.
  - Good for open areas.

Points Of Visibility



Points Of Visibility

- These can be placed by hand.
  - For maps that are not too big.
  - And that are designed by the developers / skilled modders.
- This also means you can use you own understanding of the game.
  - E.g. more nodes and a more dense visibility table in more “important” areas.

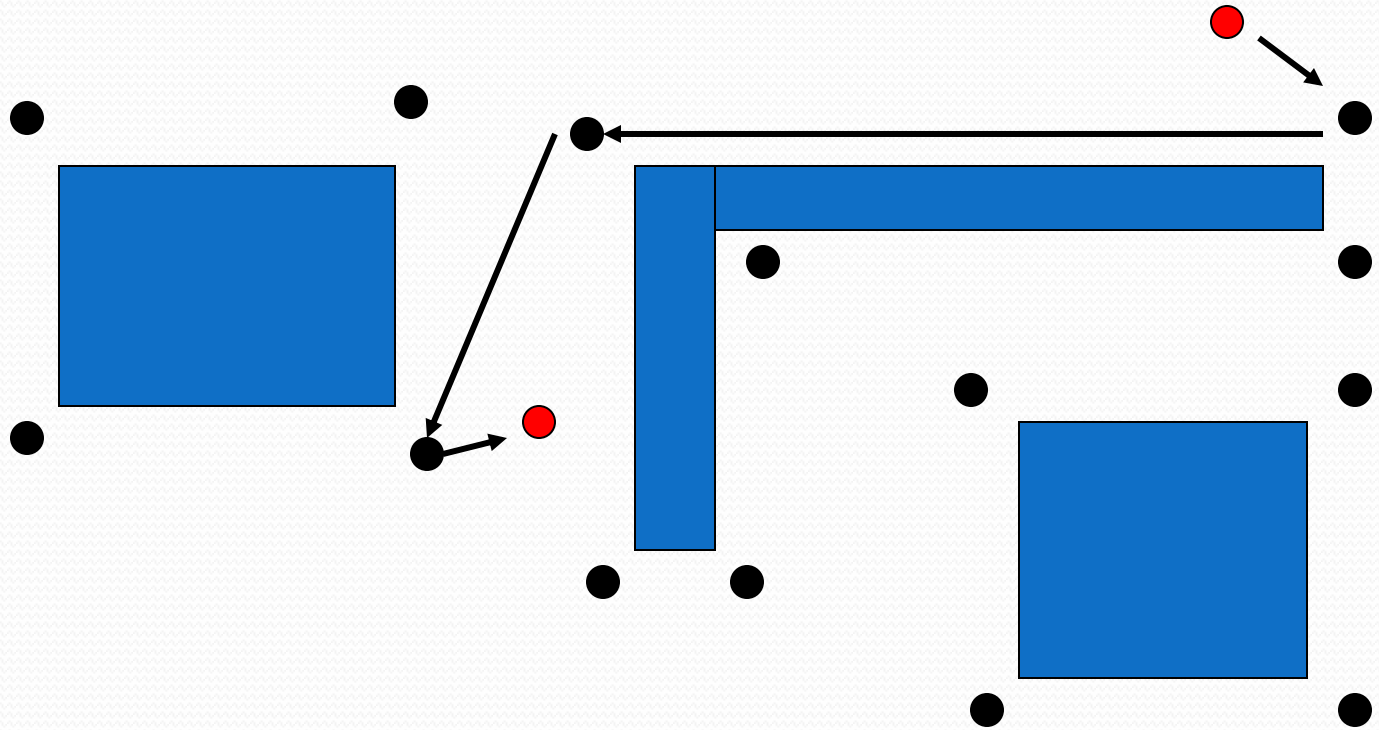
Points Of Visibility

- It is also possible to find appropriate nodes by various automatic approaches.
- Good for user-created maps.
- Can always be tinkered by hand for improved performance / path efficiency.

Points Of Visibility

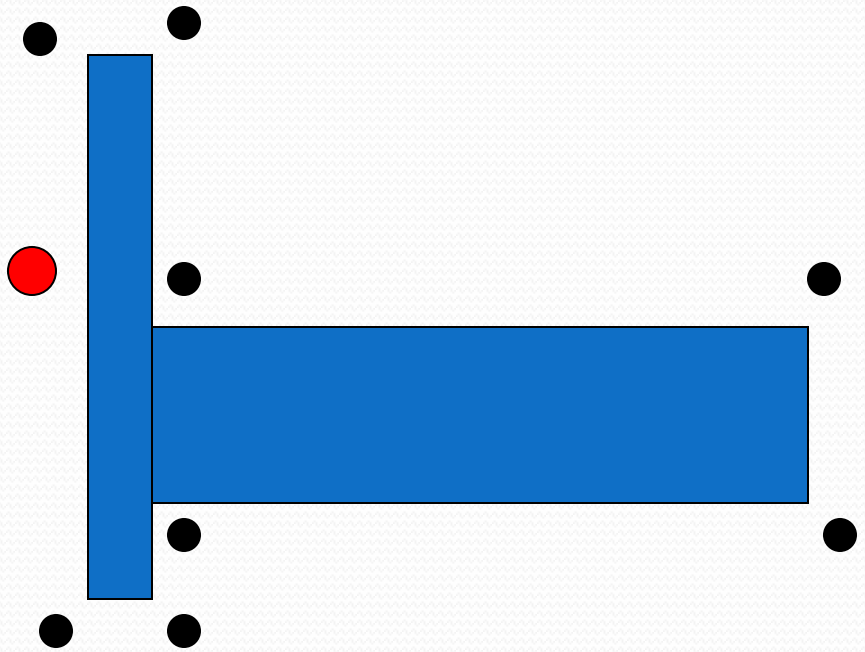
- POV can lead to a very sparse map.
- One problem here is that a character will not actually be AT a node when starting to plan a route.
  - And may not be wanting to finish at a node.
- So may have to find “closest” node.
  - Problems:

Points Of Visibility



Points Of Visibility

- You may also find that the “nearest” node is not actually visible.



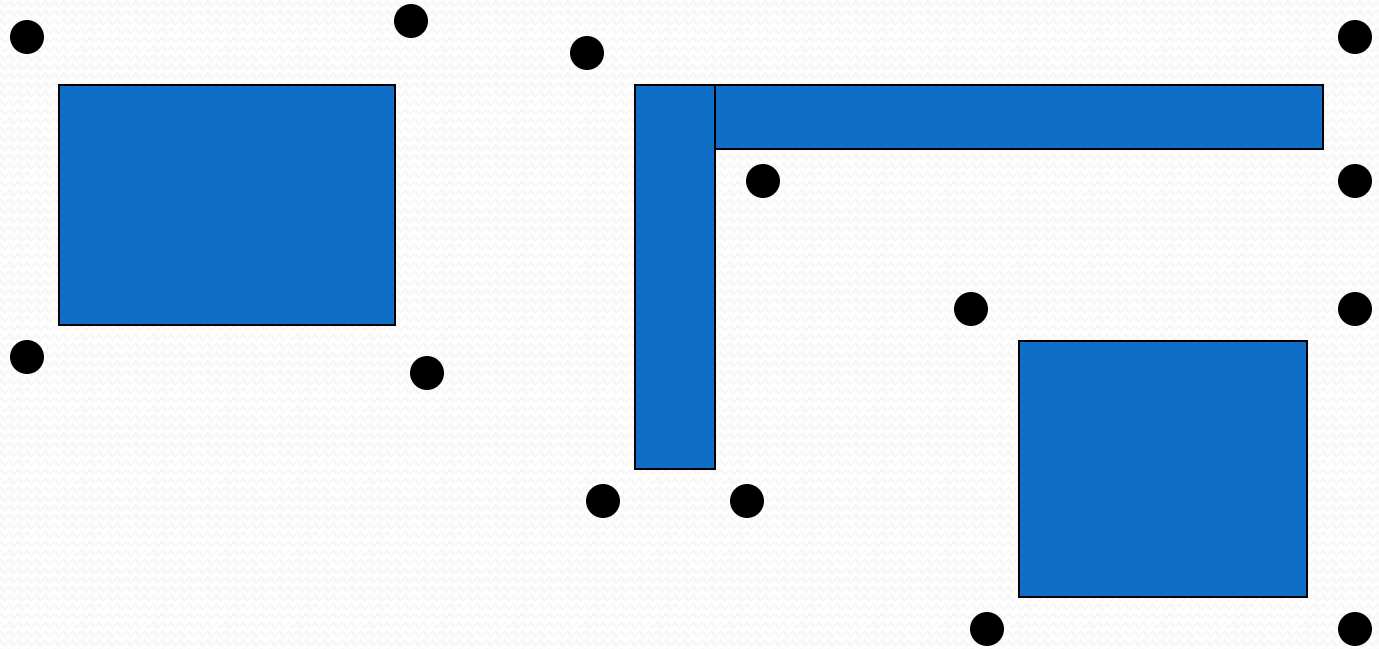
Points Of Visibility

- Adding the current position and target position as extra nodes to the visibility table is too slow to do at game-time.
- The extra processor time may be better spent on more granularity.
  - But there are a few tricks that minimise this problem.

Expanded geometry

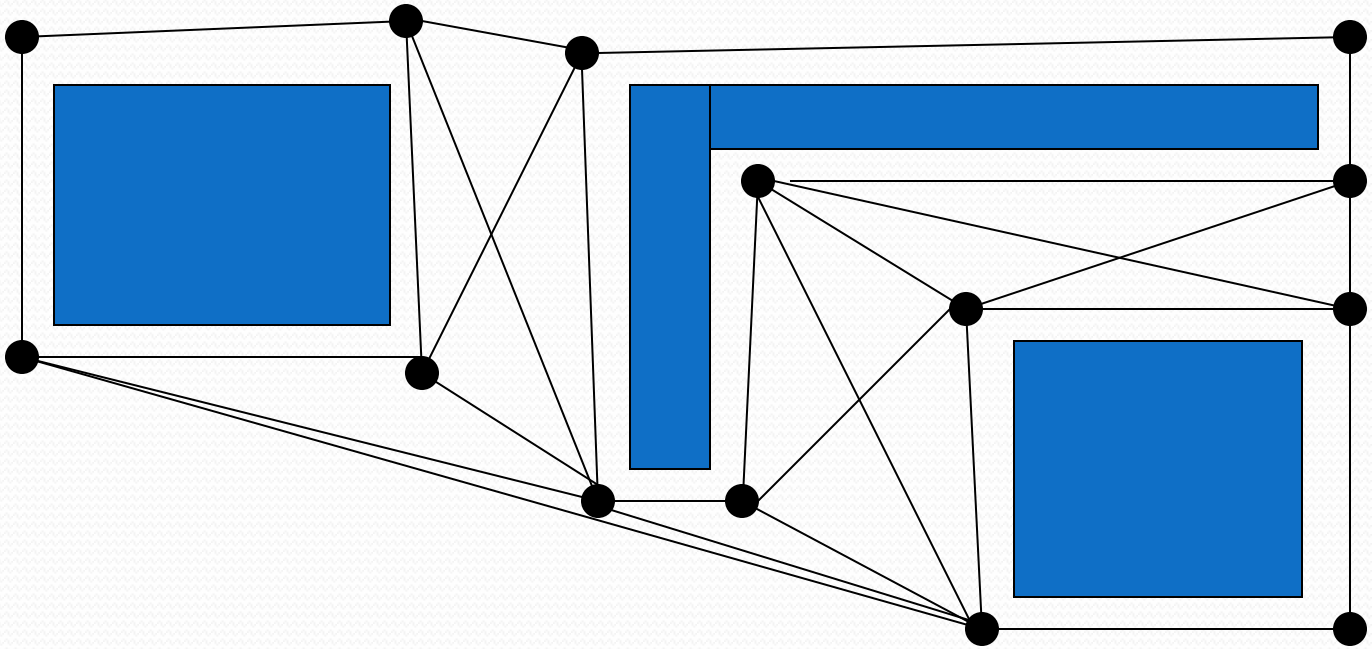
- One way to generate POV nodes automatically is to use expanded geometry.
- Just put a node slightly away from each interesting corner.

Expanded geometry



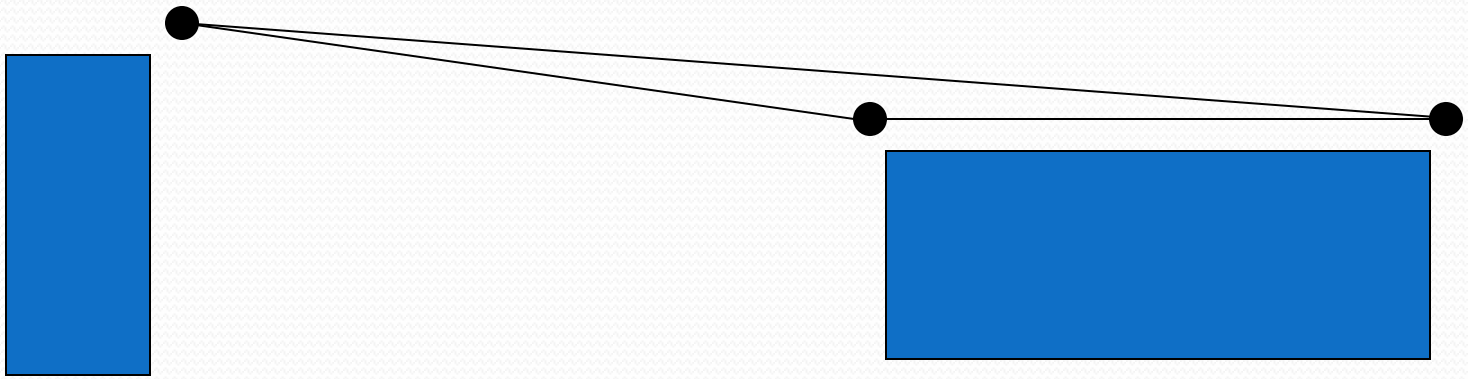
Expanded geometry

- Then check each point for visibility with each other point.



Expanded geometry

- We can “prune” paths between nodes that are not needed, as they pass too close to other intermediate nodes.

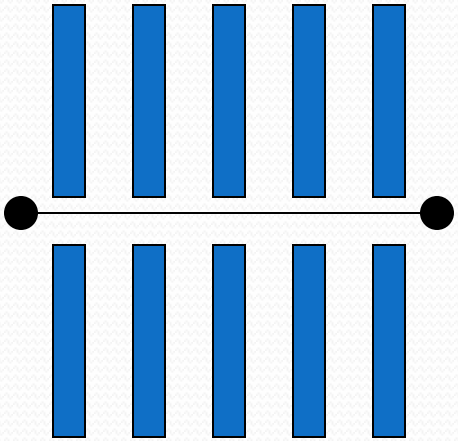


Expanded geometry

- How?
  - Node A and C are visible to each other and have a distance of 36.
  - But I can find a path A-B-C that has a distance of 36.7.
  - Remove the link between A and C.
- How close the paths need to be is up to you, but I suggest always removing identical distances. (When the three nodes are in a line.)

Expanded geometry

- But can be useful to leave in a few large jumps if expected to be useful.

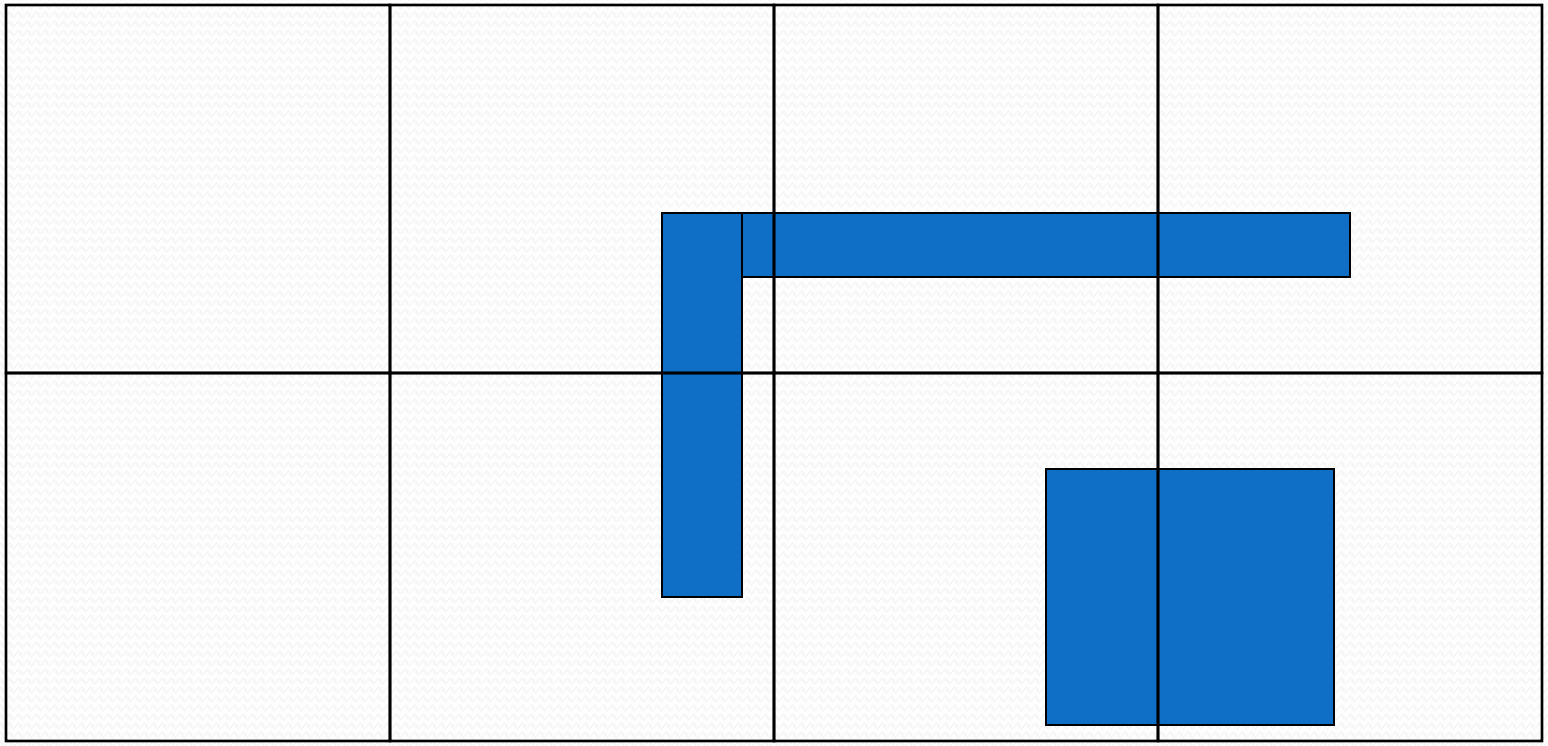


Partitioning

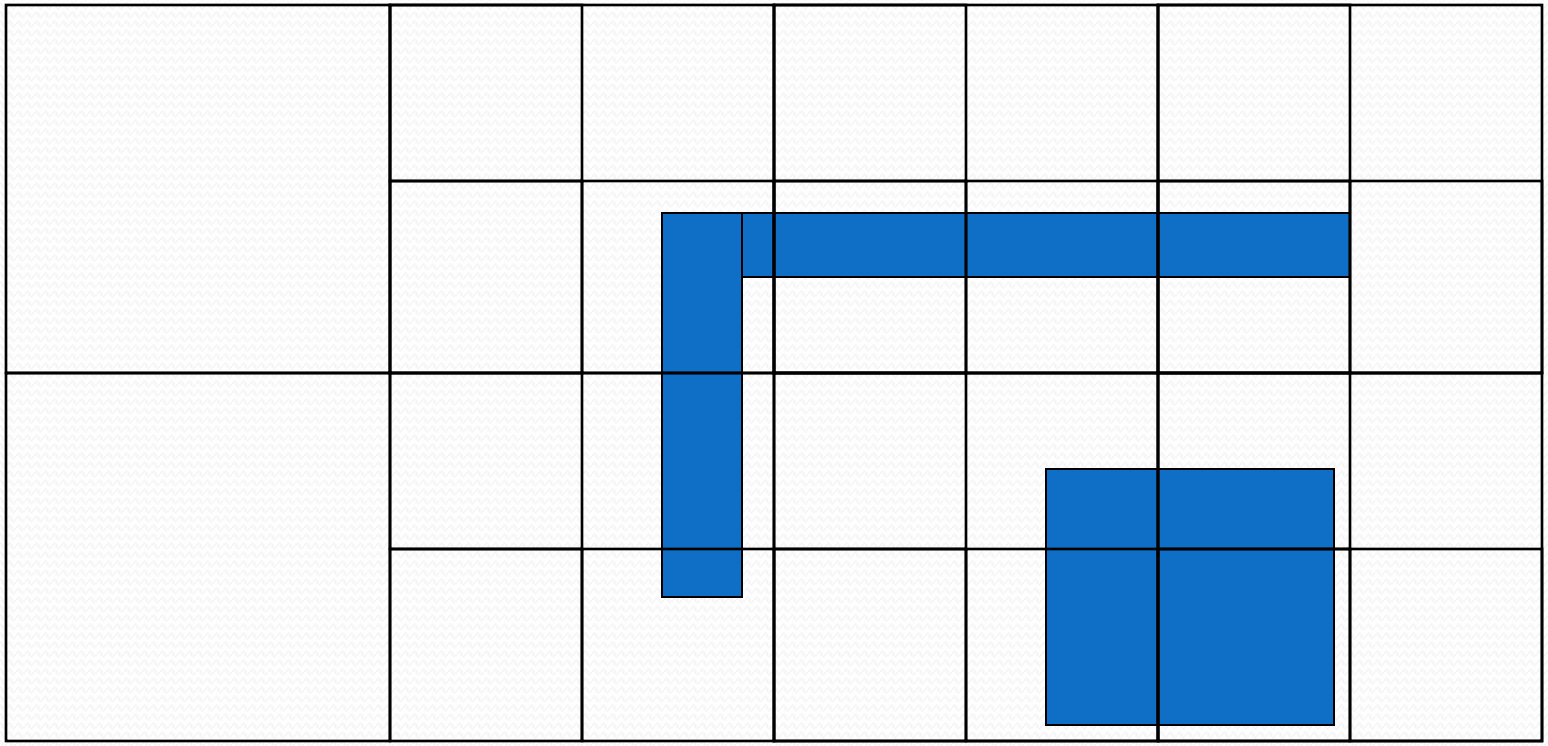
- If you have complex environments with tricky-shaped obstacles and several large open areas, another approach is to partition the space up by repeatedly dividing.

- First, divide the space up into large squares. (Or possibly rectangles/triangles)

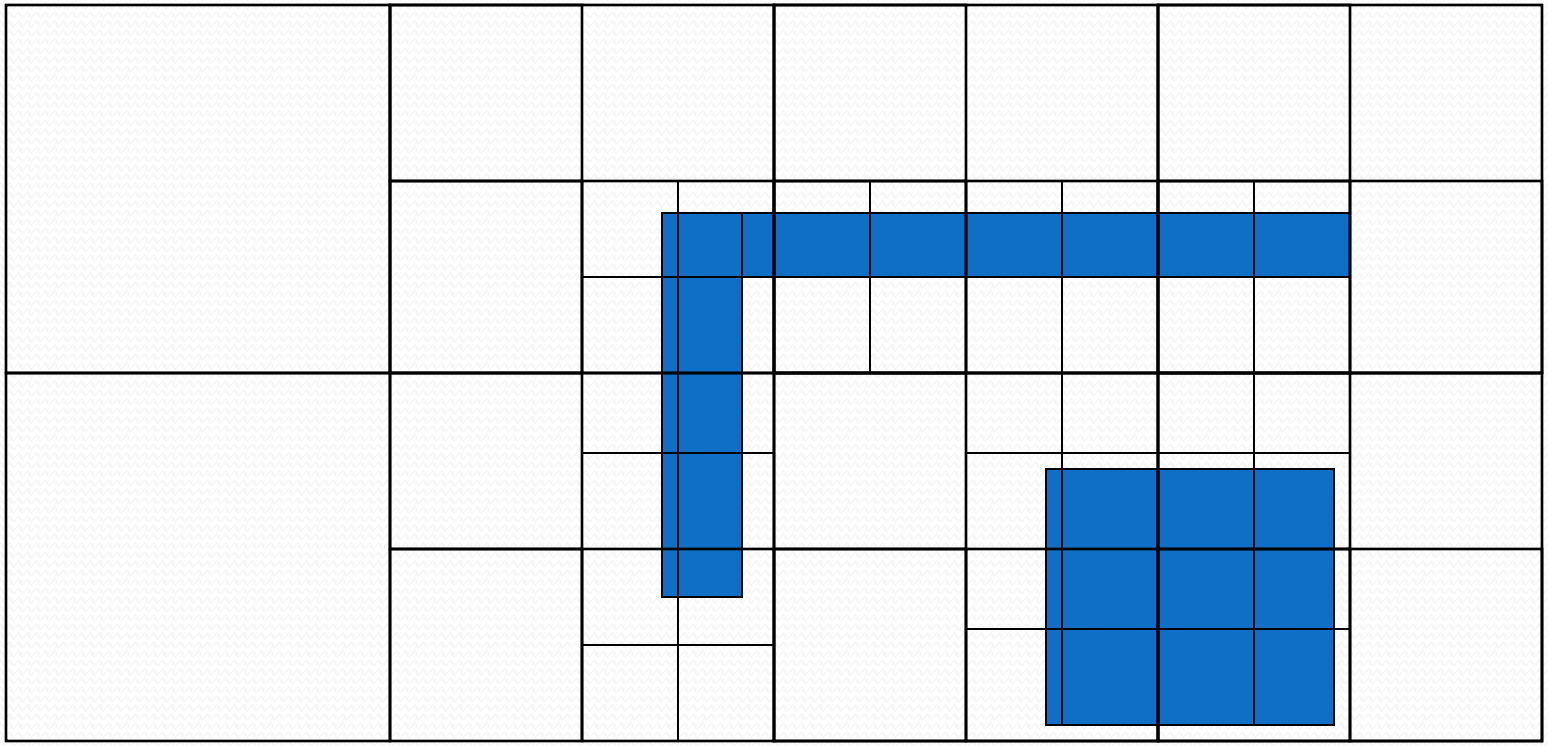
Partitioning



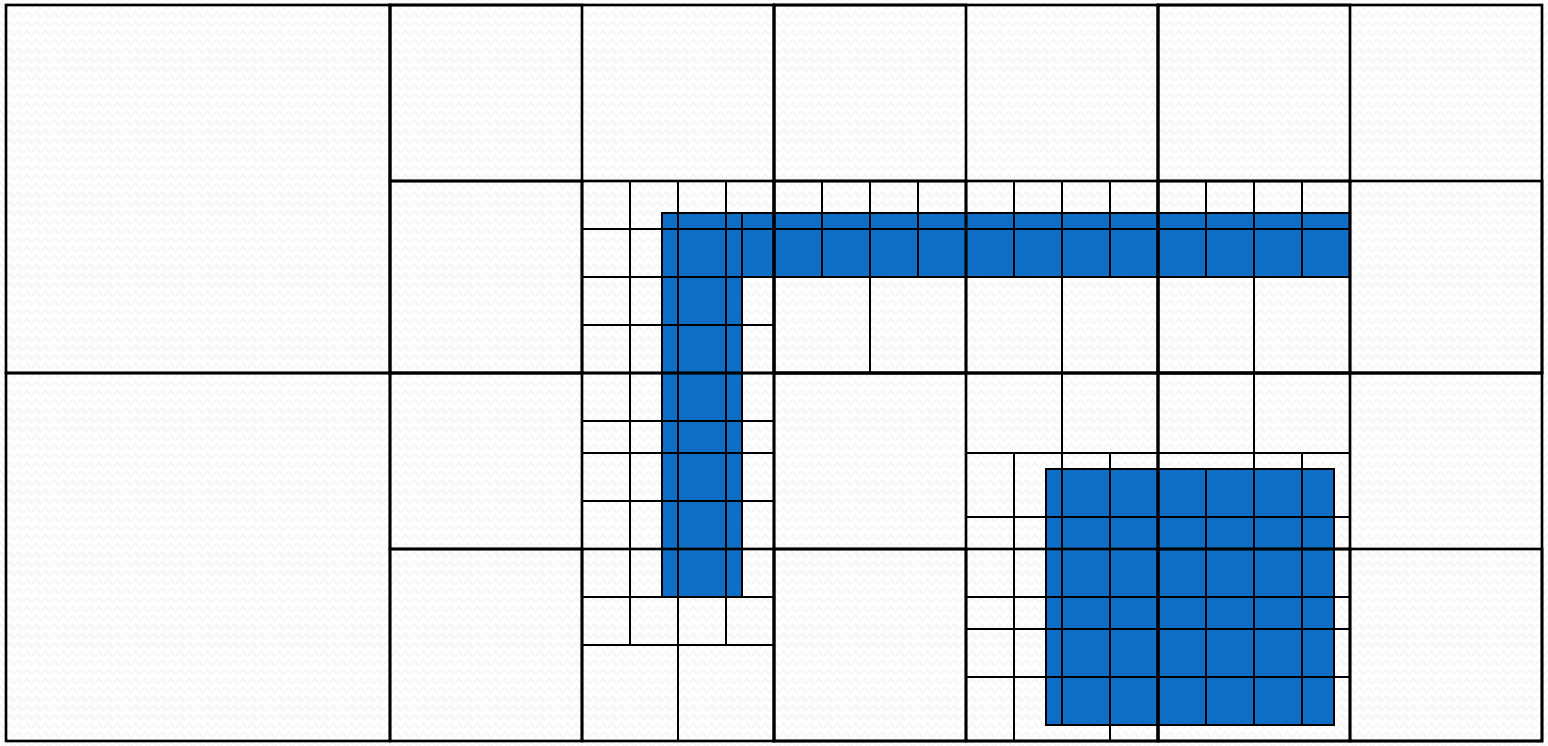
Partitioning



Partitioning



Partitioning



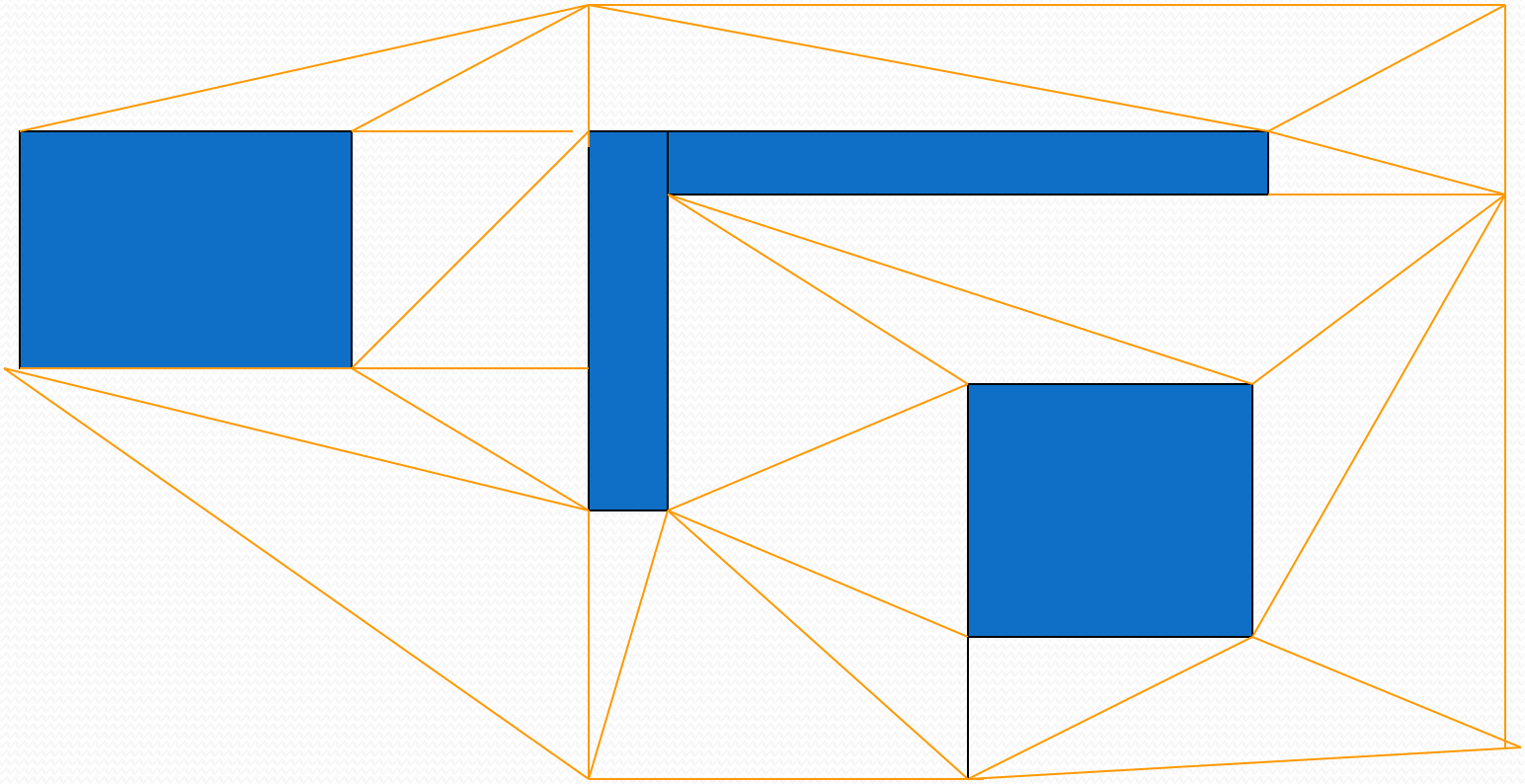
Partitioning

- Recursive function?
- Stop when the grids get too small.
- Any grid that does not intersect an obstacle:
  - Put a node at the centre of it (or the corners).
- Lots of nodes in interesting areas, but few in open spaces.
- The nodes can be stored in a quad-tree easily for fast searching.
  - This is surprisingly irritating to program, though.

Navmesh

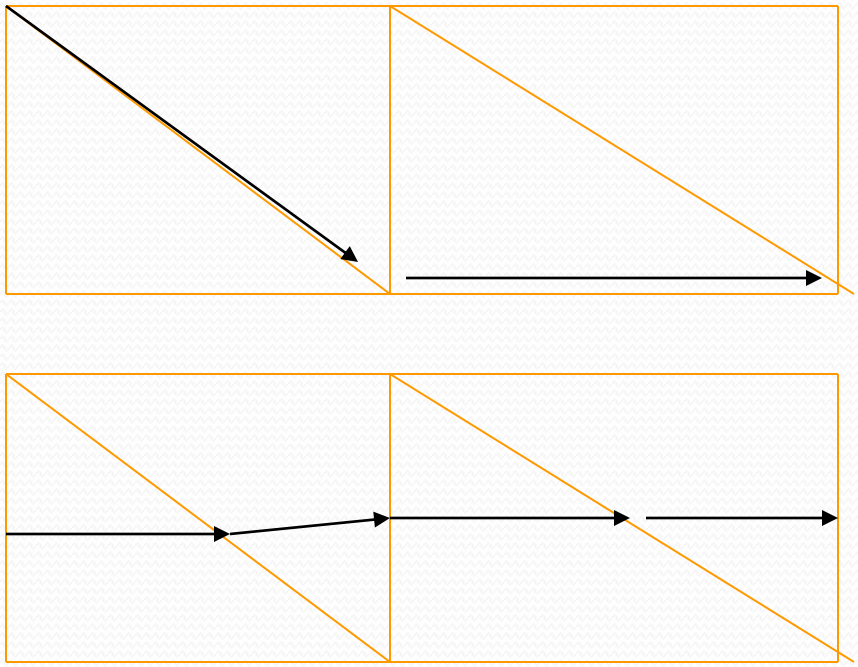
- One increasingly popular approach is to use navigation meshes.
- Split the navigable area up into polygons.
  - Usually triangles, but any convex polygon should work.

Navmesh



Navmesh

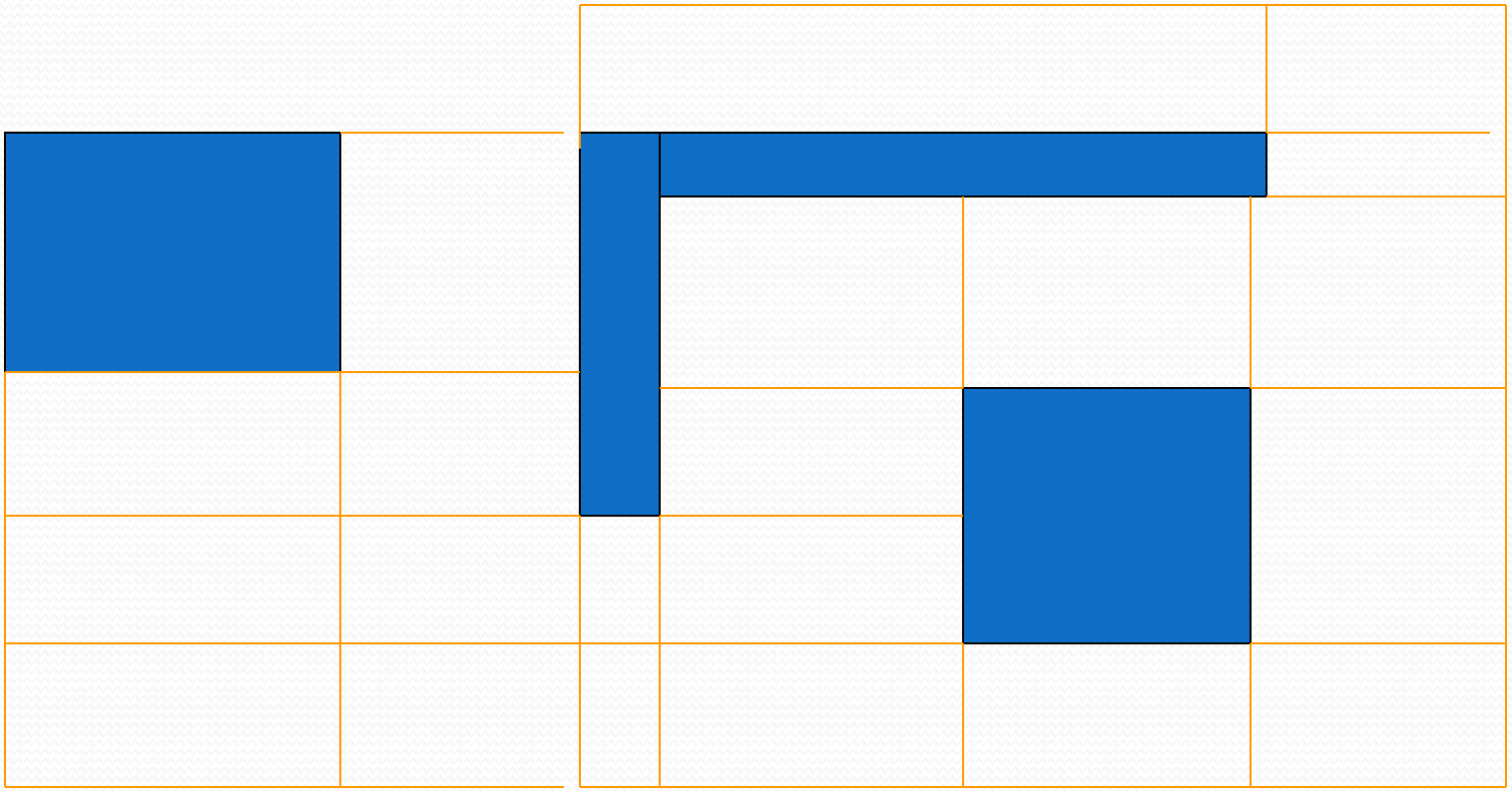
- You can then pathfind to either along the vertices, or to the mid-points of each edge.



Navmesh

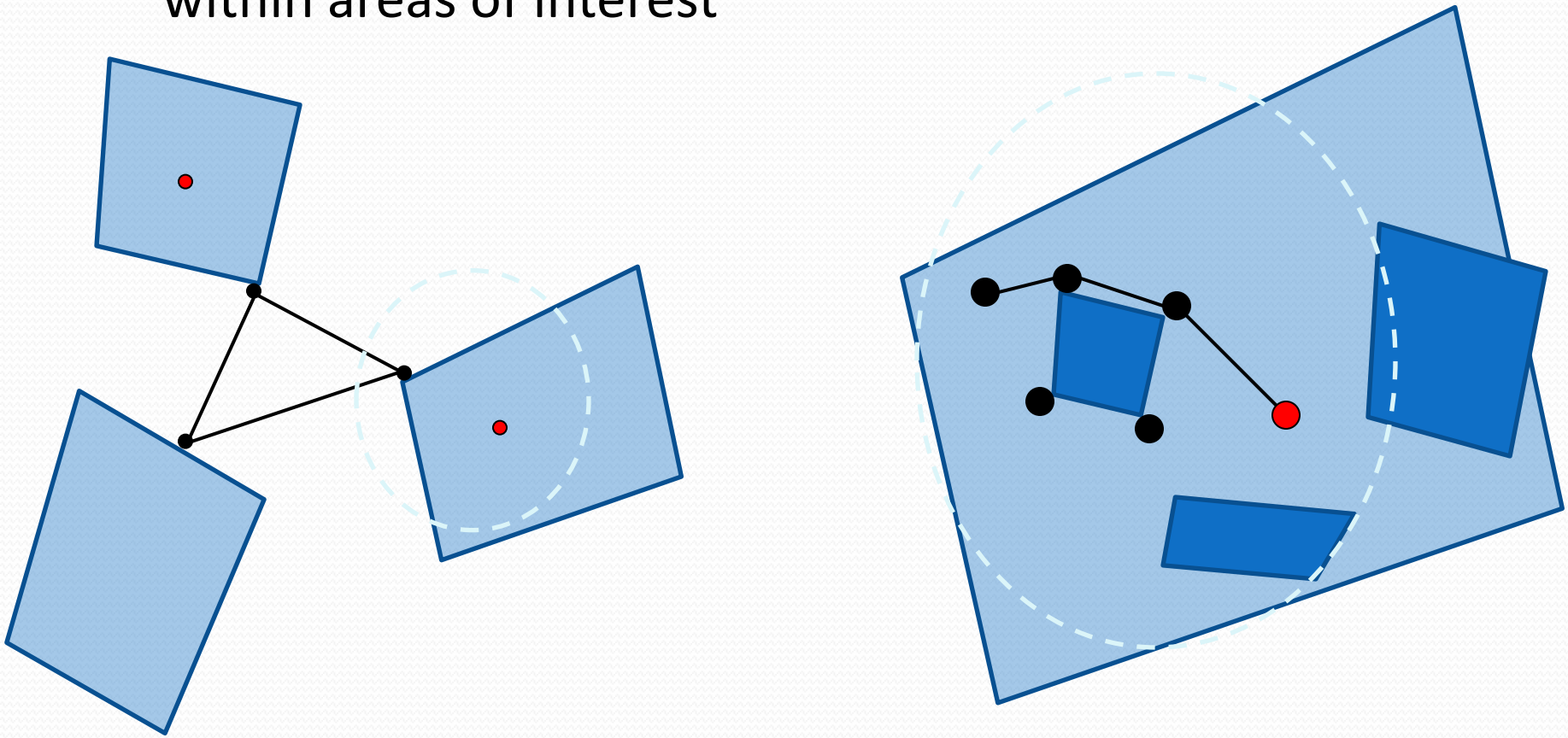
- Using the mid-points makes more sense.
  - AI doesn’t walk along the edge of a corridor.
  - Will walk through the middle of a door.
- But you can add extra vertices to avoid these problems.
- For a grid-aligned map with lots of internal rooms, you may find it easier to use rectangles.

Navmesh



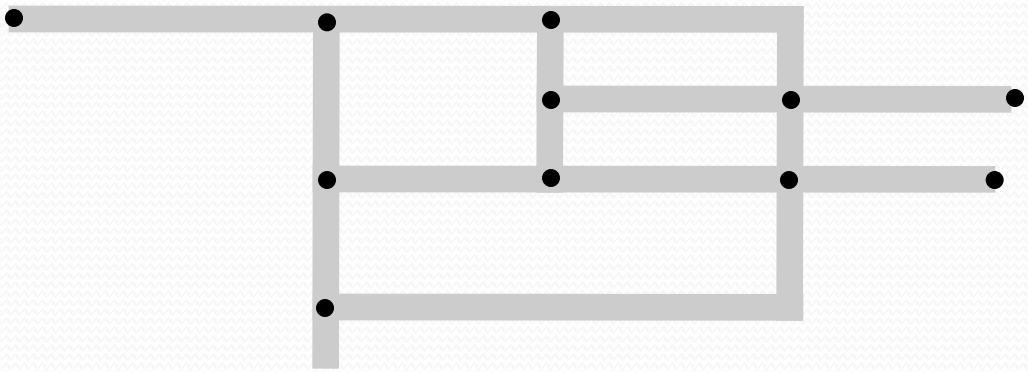
Hierarchical Maps

- Use maps at different scales to find routes between / within areas of interest



Roadmaps

- Treat the roadmap as a graph and waypoint the intersections
  - NB *NOT* the corners!



Summary

- Nodes
- Visibility tables
- Generating nodes
  - Rectangular grid
  - Points of visibility
  - Expanded geometry
  - Partitioning
  - Navmesh

Reading

- Programming game AI by example.
  - (Brief).
- Game programming Gems
  - Simplified 3D movement and pathfinding using navigation meshes
- AI Game programming Wisdom
  - Strategic and tactical reasoning with waypoints.
- http://theory.stanford.edu/~amitp/GameProgramming/MapRepresentations.html

Reading

- Terrain analysis
  - Liden, L. (2001). The use of artificial intelligence in the computer game industry.
  - Liden, L. (2002). Strategic and Tactical Reasoning with Waypoints. AI Game Programming Wisdom. S. Rabin. Massachusetts, Charles River Media: 211-220.
  - Reich, A. J. (1995). An Efficient Representation of Spatial Data For Terrain Reasoning By Computer Generated Forces. ELECSIM95.
  - Petty, M. D. F., Robert W. Mukherjee, M (1999). A terrain reasoning algorithm for defending a fire zone. Information and security. **3**.