Terrain Analysis

EECS 370 Spring 2017

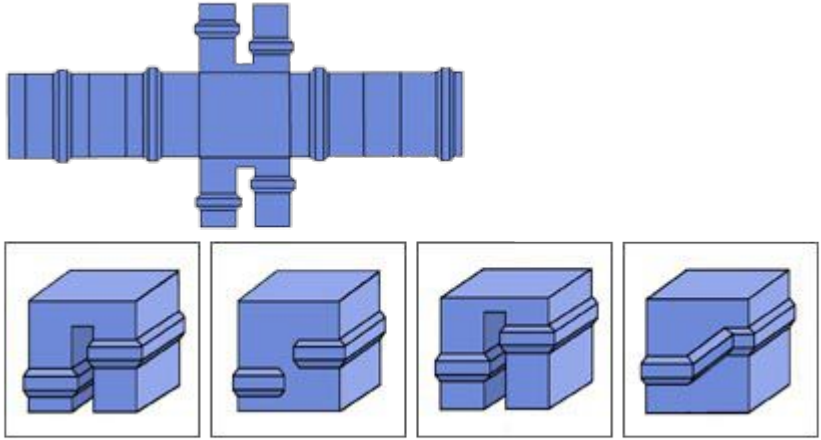
Overview

- Space is hard
- Path-finding
- Position-finding
- High-level terrain analysis, planning

Spatial Reasoning Is Really Hard

- Humans have incredible perceptual and motor skills dedicated to handling parts of it
- Humans appear* to have rich representations of space and shape at multiple levels of description
- Simplification: we can try to construct discrete underlying representations

* we don’t really know the brain very well



For Developers

“The first of many hurdles to cross is to prevent the game from freezing every time we path some unit across the screen. This ‘game freeze’ can be considered the plague of pathfinders. It’s a game killer, and one of the most significant problems we will need to solve until we all have 5-trilliahertz computers with googles of memory”

- Dan Higgens (*Empire Earth* team from Stainless Steel Studios, Inc.)

For Players

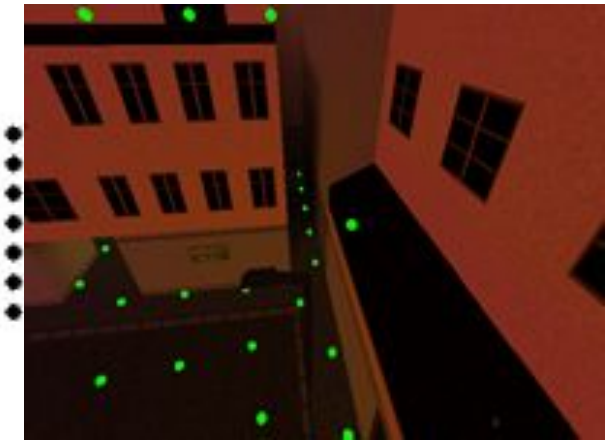
“Managing that army can be a bit annoying at times, though, because of AoM’s errant pathfinding. On more than one occasion, I’d try to send my units to a specific spot on the map and they’d end up on a hill overlooking the spot where I wanted them to be. Other problems include units getting trapped between rows of bushes, units ‘jerking’ this way and that as they move, and units hugging a cliff instead of simply walking down the road in front of them (which results in them moving much slower than they should).”

- Age of Mythology review, *PC Gamer,* 2002

General Approach

Turn a continuous, infinite space into a discrete, finite one

- Simplify our representation of space
  - sometimes hand-constructed by level designers
  - sometimes generated, when PCG or user-made levels are involved

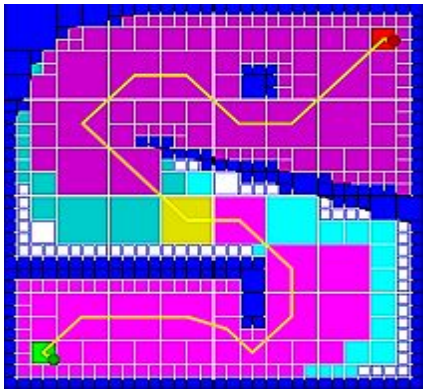


http://www.gamasutra.com/view/feature/131447/terrain_reasoning_for_3d_action_.php?page=2

Families of Spatial Solutions

- Tiles
- Waypoints
- Quad trees
- Navigation meshes

Making Graphs



Tiles

- Variations
  - rectangular or isometric, depending on perspective
  - hexagonal grids model distances traveled on diagonals more accurately
- Tradeoffs
  - very simple
  - uniform resolution can waste storage on uninteresting regions of space
  - low-res tiles can be transparent to players (pros and cons)

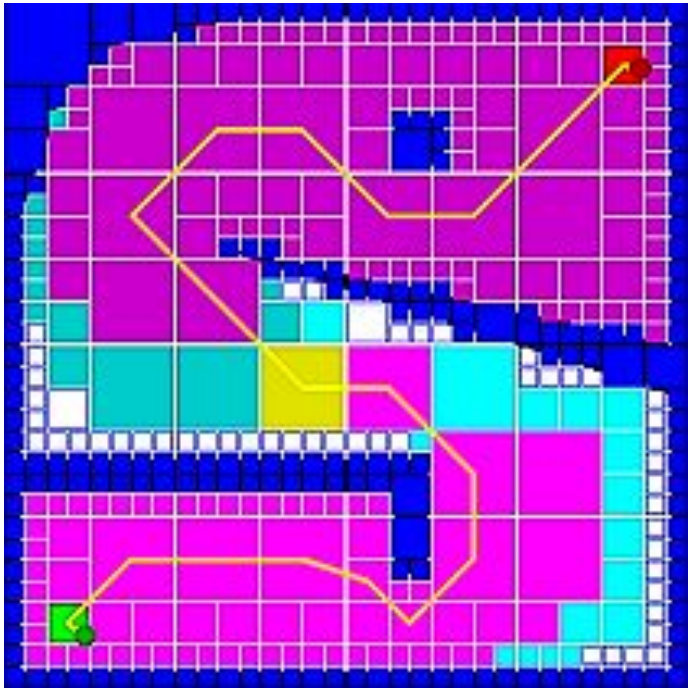
Tile Size == Unit Size

Tile Size << Unit Size

Quad Trees

Carve up space according to where objects *aren't*

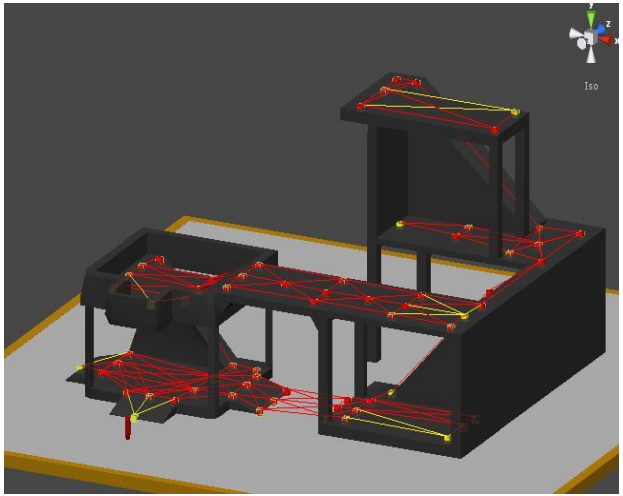
- Stop conditions:
  - uniform contents
  - recursion depth limits
- Characteristics:
  - variable resolution
  - can be generated automatically



Waypoints

Annotate terrain with hand-selected places that movable entities can be in

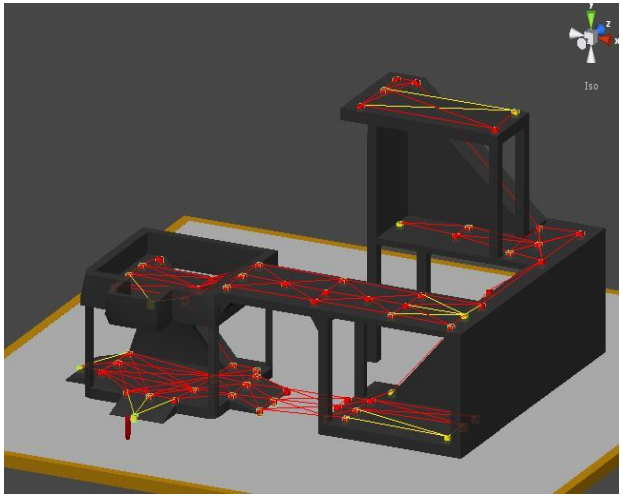
- Adjacency relationships between waypoints indicate ways to move from one to the other (including travel time)
- Additional annotations can be used to indicate other properties
  - whether line-of-sight exists
  - part of a room or some interesting location
  - environment conditions, such as light/dark, types of movement required



Waypoints

Tradeoffs:

- Can analyze to derive many useful tactical properties
- Often hand-generated or hand-tweaked
  - density:
    - high density ⇒ slower pathfinding
    - low density ⇒ characters zigzag



Nav Meshes

- Convex traversable polygons
- Similar to waypoints, but you know all the areas enclosed are totally traversable
- Tradeoffs:
  - efficient for variable-density worlds
  - more flexible than waypoints
  - more robust to interference
  - still requires annotation

Complex Solutions

- Many games use a combination of big strategies and smaller tricks for handling exceptions and emergencies
- Scaled to complexity
  - if a creature's life expectancy is short, it can be cheaper and less clever
  - a creature that needs to live longer needs to path smarter
    - fellow soldiers, escortees
  - player-issued commands vs. NPC movement
- Predictability
- Parallelism

Path-finding is... easier

- A* is the cheap and easy way to do path-planning
- It has a lot of problems if you want realistic movement
- There’s more to moving through space than just path-finding

Dijkstra’s Algorithm

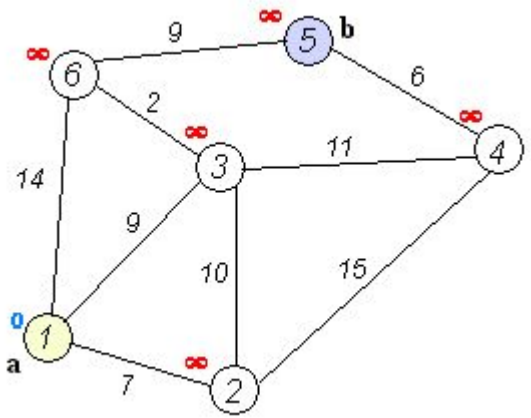
Dijkstra’s original algorithm (1956) was pretty good:

- $O(|V|^2)$
- $O(|E| + |V| \log(|V|))$ with a min-priority queue (1984)

It’s also super easy to implement!

1. queue up all neighbors of a node*
2. traverse, setting the distance to the node to **min(***current, candidate***)**
3. keep track of shortest distance to *B* and stop when impossible to beat
4. keep track of the shortest path between *A* and each node

* put them in increasing order of distance and you’ve got the improved version



https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

A*

So Dijkstra's is pretty dope, but there's a problem:

- it searches the entire space of options more or less evenly until it finds the goal node

What if we tried to guess order ahead of time?

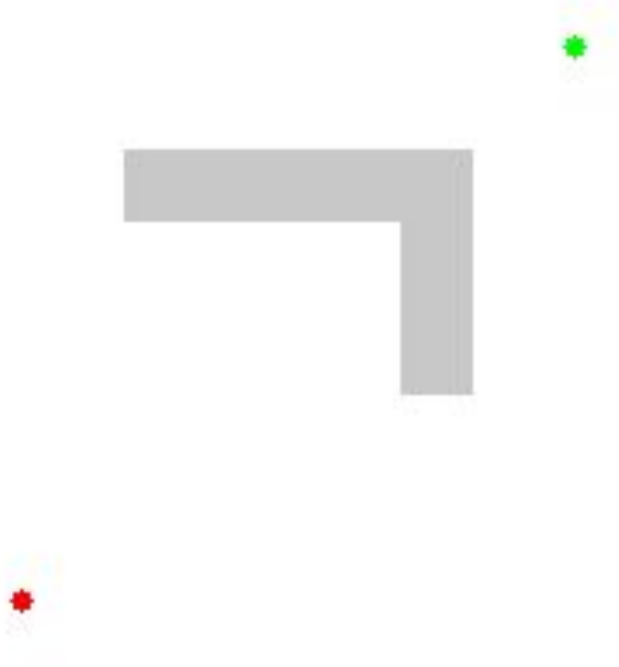
- Dijkstra's priority queue is sorted by edge length
- A* sorts by length, plus a heuristic

A*

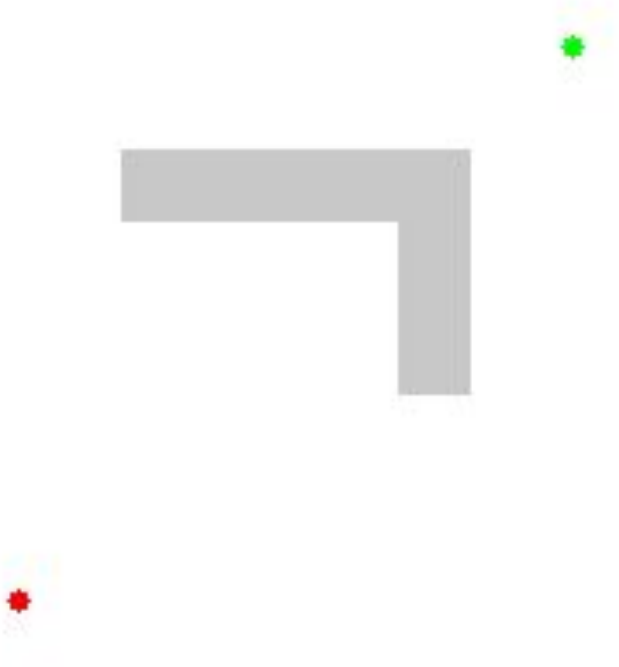
- The easiest heuristic to use is euclidean distance
  - meaning that we search nodes in the direction of the goal first
  - you know, like people tend to do
- Caveat emptor: A* is optimistic; its heuristic can never* overestimate the distance between a candidate and a goal
- Notice that Dijkstra's is just A* where every node's heuristic function returns the same value

* double caveat-we can kind of relax this requirement, but generally don't

A*



https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

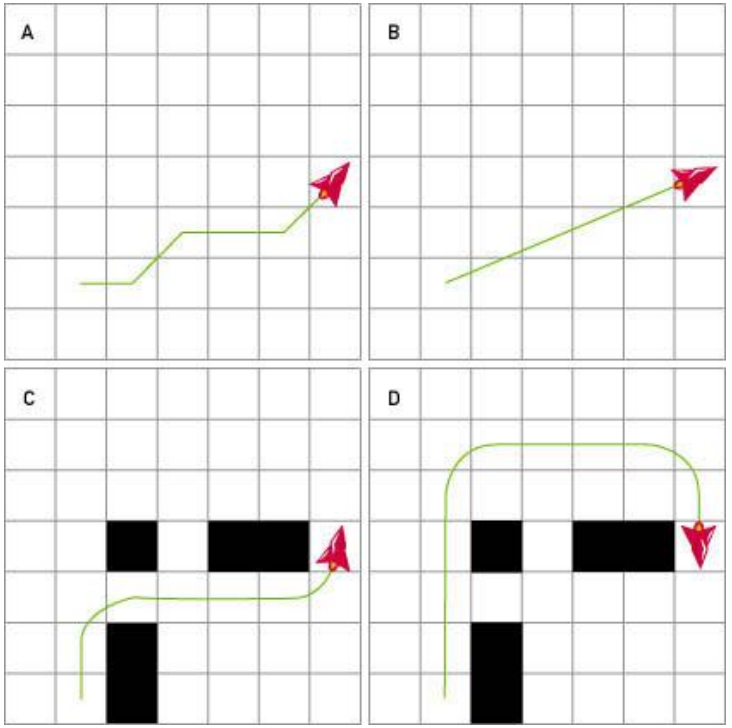


https://en.wikipedia.org/wiki/A*_search_algorithm

Pathfinding Can Involve Multiple Constraints

Problems with Pathfinding

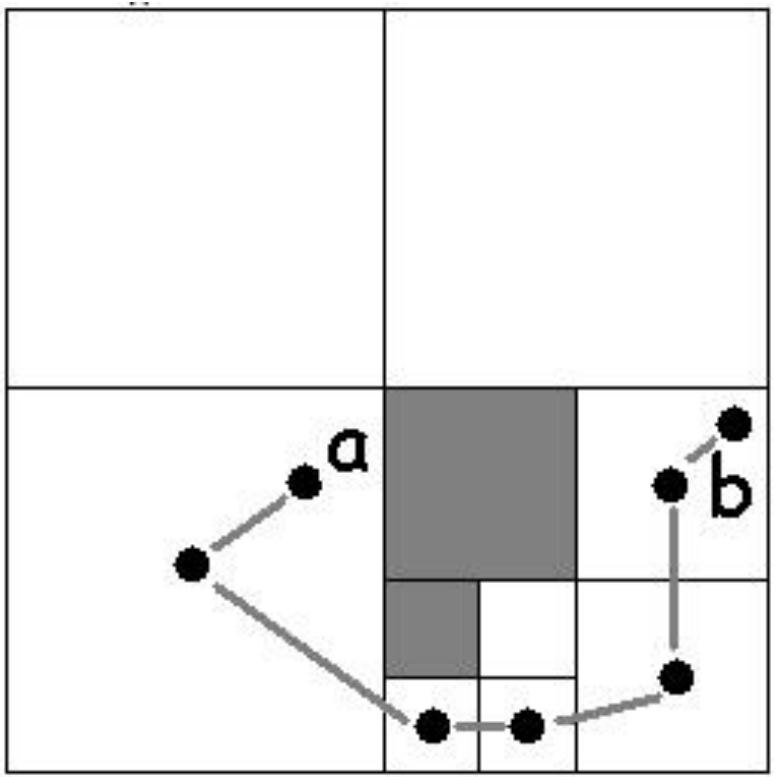
- Zig-zagging
- Smooth turning
- Turning radius



http://www.gamasutra.com/view/feature/131505/toward_more_realistic_pathfinding.php

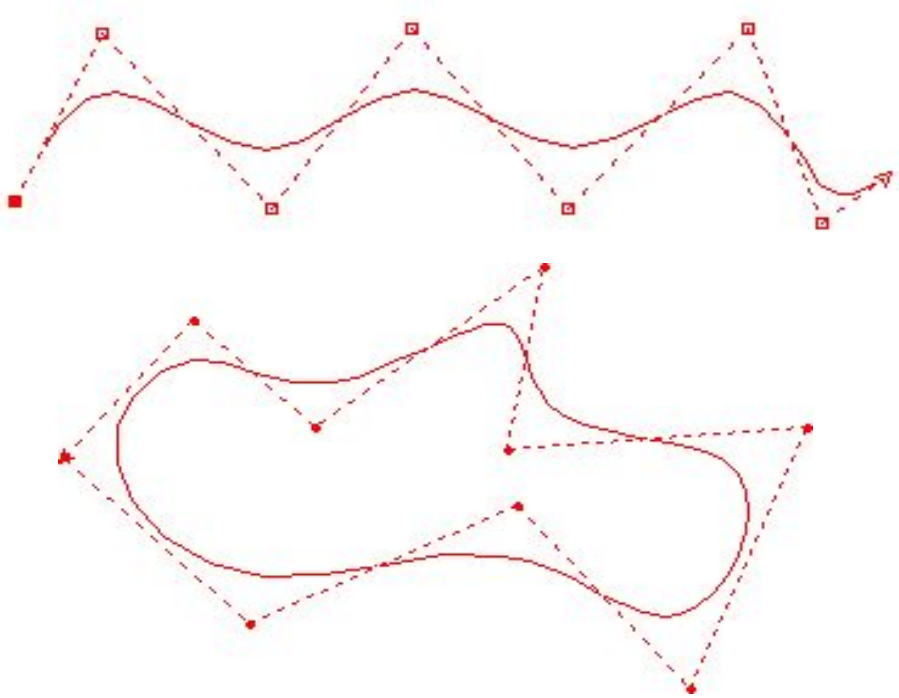
Problems with Pathfinding

- Confusing inefficiencies



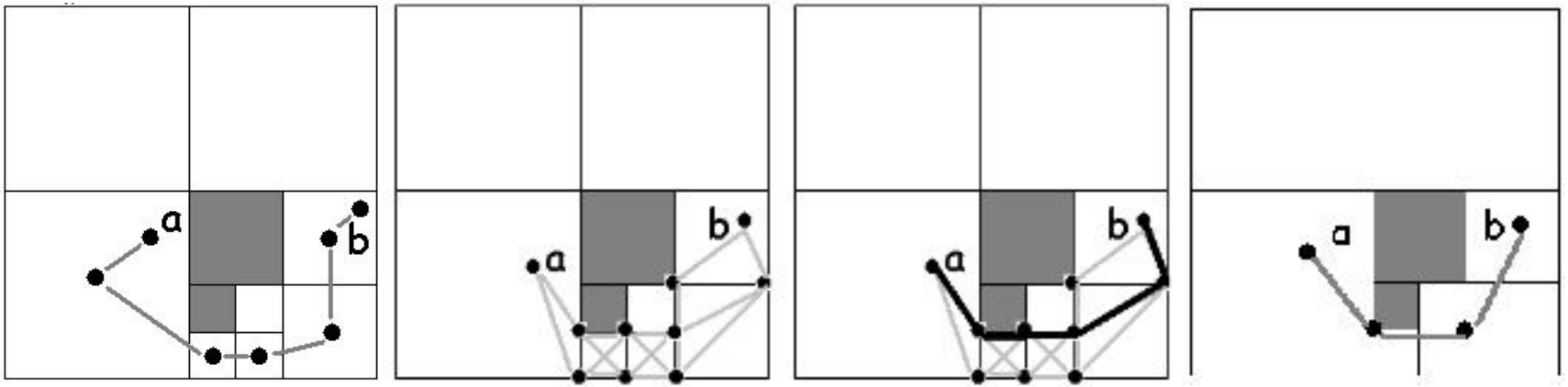
Cleaning up Paths - Splines

- Splines provide a smooth approximation to the original path
- Problem: waypoints were there for a reason



Cleaning up Paths - Rubber Banding

- Figure out the best corners to travel to



Working in Parallel

- Many systems use multiple layers of representation
- Plan the route on a macro scale (region-level)
- Work out how to navigate through local area
- Rubberbanding to make paths efficient
- Detecting when something has gone wrong

Position-finding

- There are lots of ways of prioritizing space
  - A* and other path-finding algorithms rely on weighting edges between nodes
- Representations of space
  - we got graphs
  - hierarchical representations of space (macro vs. micro)
  - territory and influence

Prioritizing Space

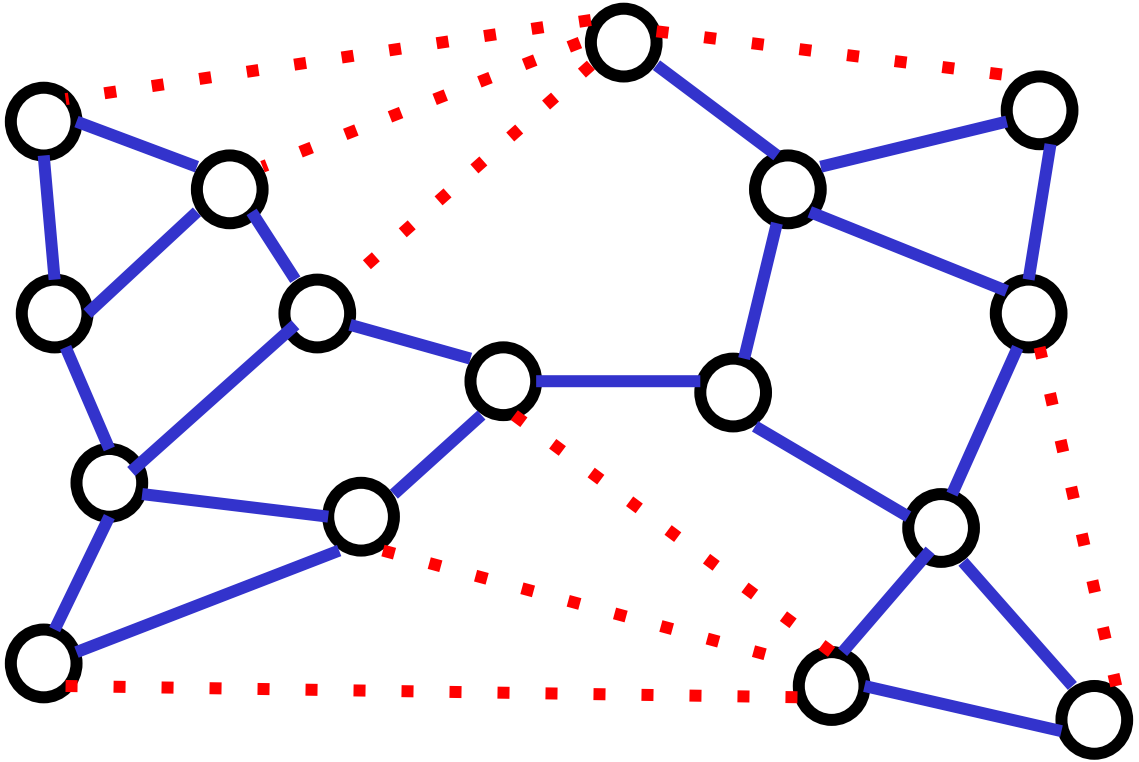
Many problems in games involving finding places

- Where should I build cities?
- How can I cut crime in my city?
- Where could I ambush them?
- Where would be a good place to hide?
- What would be the most profitable place to put a ride?

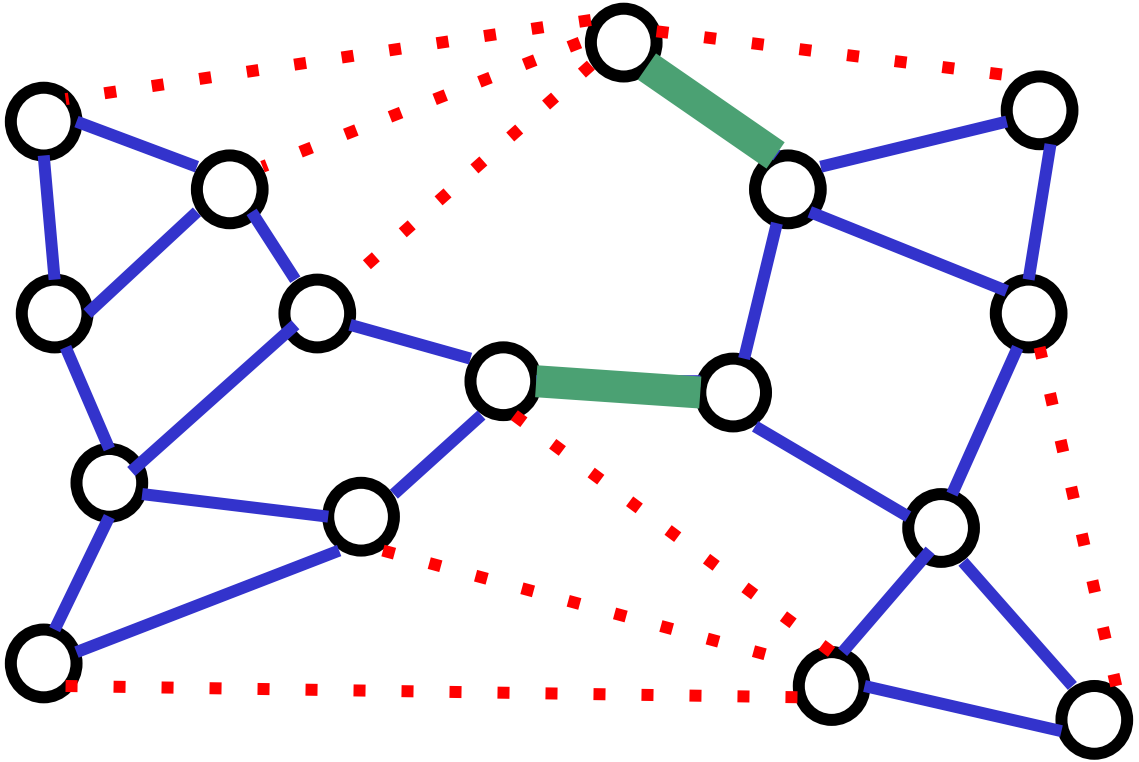
Position-finding = finding locations that satisfy (or optimize) particular criteria

- Multiple constraints can be involved
- What algorithms are used depend on the available representations

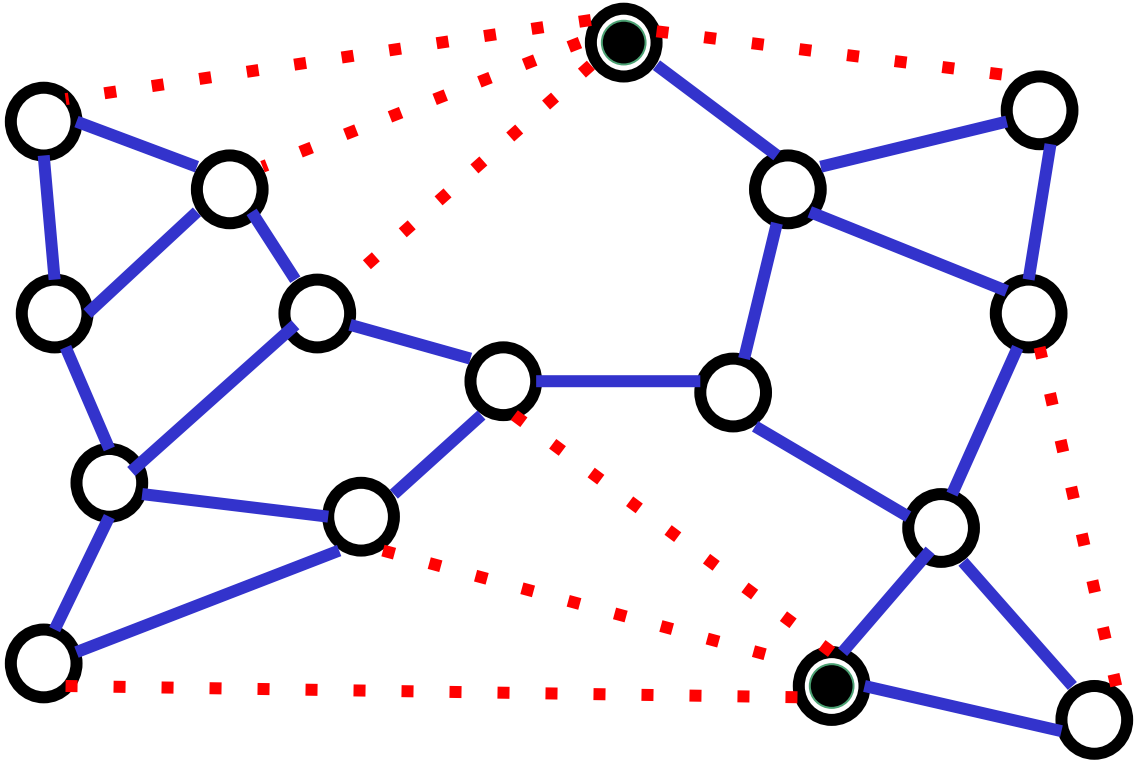
Example: Tactical Positions



Example: Choke Points



Example: Snipers



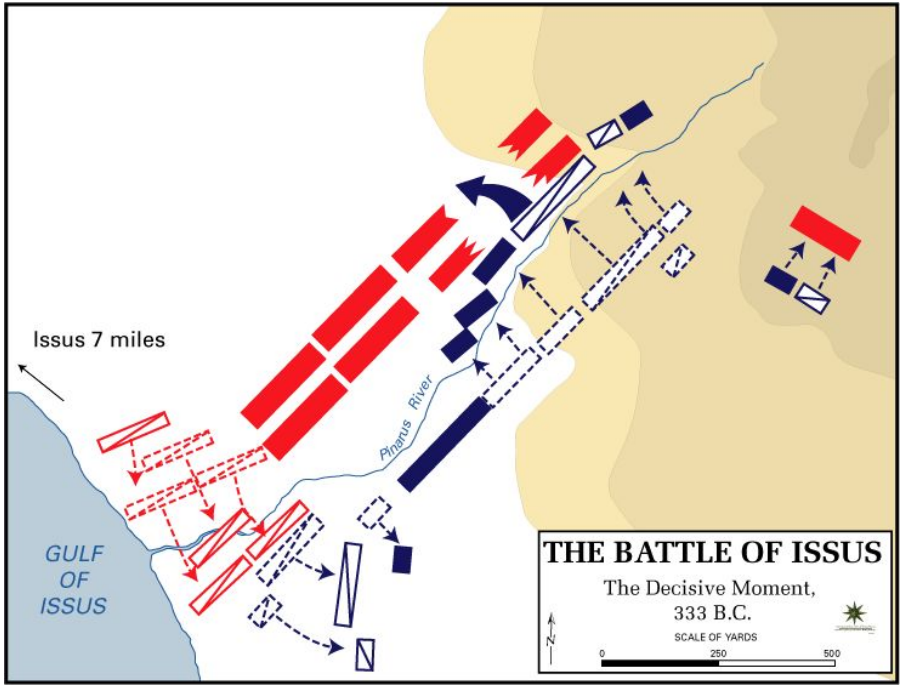
Influence Maps

Influence Maps

Planning

- Tactics! (also strategy)
- Decision Trees
- STRIPS/GOAP
- Faking Personality

These things are good because they make NPCs seem less dumb (I did not say smarter).



https://en.wikipedia.org/wiki/List_of_military_tactics#Small_unit_tactics

Questions?