Advanced Pathfinding

Technical Game Development II

Professor Charles Rich
Computer Science Department
rich@wpi.edu

With material from: Millington and Funge, *Artificial Intelligence for Games*, Morgan Kaufmann 2009 (Chapter 4) and Buckland, *Programming Game AI by Example*, Wordware 2005 (Chapter 5, 8).

A* Pathfinding Search

- Covered in detail in IMGD 3000
- See pseudo-code and links to reference code at http://web.cs.wpi.edu/~gogo/courses/imgd3000/slides/imgd3000_08_AI_A_Star.pdf
- **Basic A*** is a *technical requirement* for final project
  - you may use any reference code as a guide, but not copy and paste (cf. academic honesty policies)

Practical Path Planning

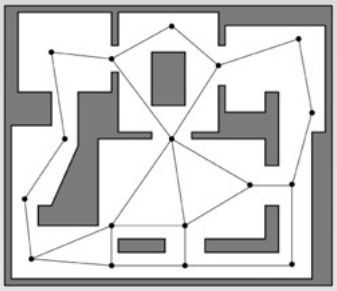
- Just raw A* is often not enough
- Also need:
  - navigation graphs
    - points of visibility (POV)
    - navmesh
  - path smoothing
  - compute-time optimizations
  - hierarchical pathfinding
  - special case methods

Navigation Graph Construction

- Tile (cell) based
  - very common, esp. if env't already designed in squares or hexagons
  - node center of cell; edges to adjacent cells
  - each cell already labeled with material (mud, etc.)
  - *downside:*
    - modest 100x100 cell map
    - 10,000 nodes and 78,000 edges
    - can burden CPU and memory, especially if multiple AI's calling in

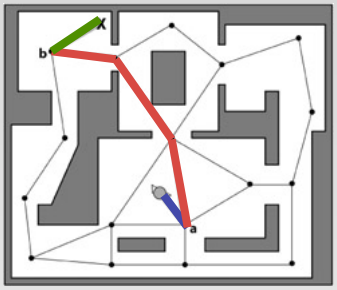
*Rest of lecture is a **survey** about how to do better...*

Point of Visibility (POV) Navigation Graph



- Place graph nodes (usually by hand) at important points in env't
- Such that each node has line of sight to at least one other node

POV Navigation

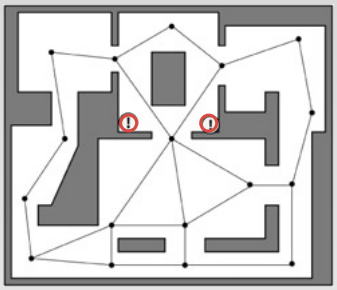


- find closest *visible* node (a) to current location
- find closest *visible* node (b) to target location
- search for least cost path from (a) to (b)
- move to (a)
- follow path to (b)
- move to target location

note "backtracking"

DEMO (COARSE)

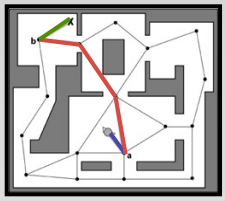
Blind Spots in POV



- No POV point is visible from red spots!
- Easy to fix manually in small graphs
- A problem in larger graphs

DEMO (COARSE)

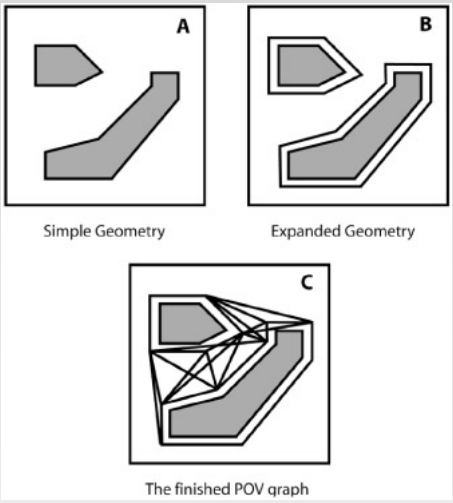
POV Navigation



- Obvious how to build and expand
- ***Downsides***
  - can take a lot of developer time, especially if design is rapidly evolving
  - problematic if random or user generated maps
  - can have "blind spots"
  - can have "jerky" paths
- ***Solutions***
  1. automatically generate POV graphs
  2. make finer grained graphs
  3. path smoothing

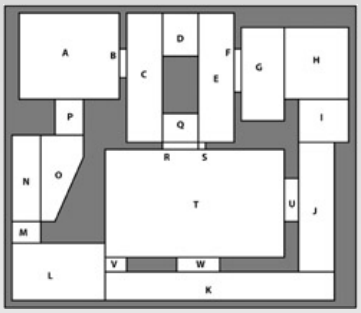
Automatic POV by Expanded Geometry

1. expand geometry by amount proportional to bounding radius of agents
2. add vertices to graph
3. prune non line of sight points

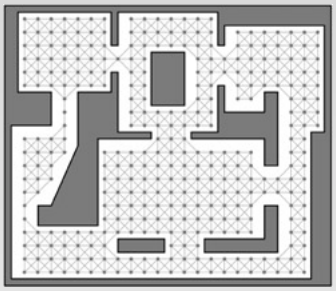


NavMesh

- network of *convex* polygons
  - why convex?
  - guaranteed to be path from any point to any point inside
- very efficient to search
- can be automatically generated from polygons
- becoming very popular



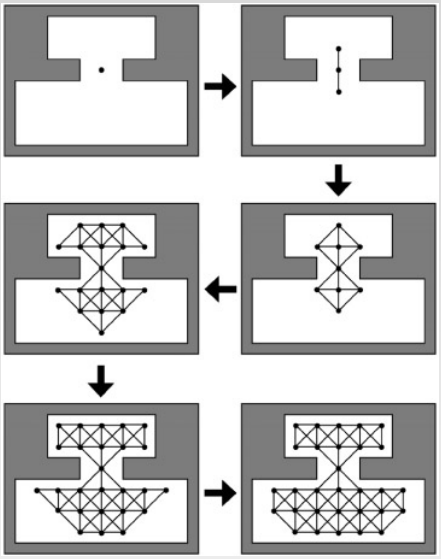
Finely Grained Graphs



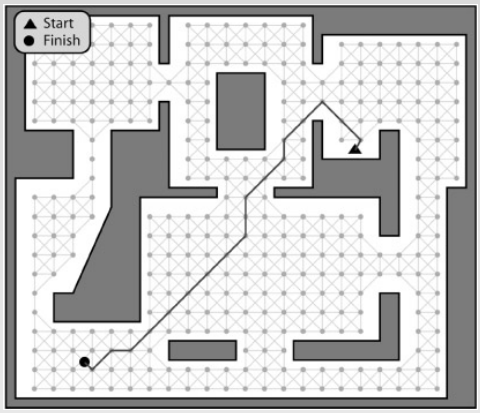
- Improves blind spots and path smoothness
- Typically generate automatically using “flood fill”
- Back to similar performance issues as tiled graphs

Flood Fill

• same algorithm as in “paint” programs

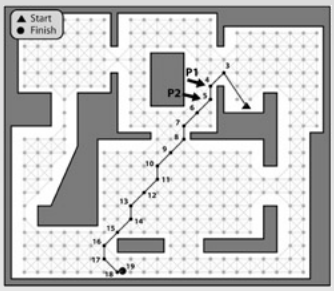


Path Finding in Finely Grained Graph



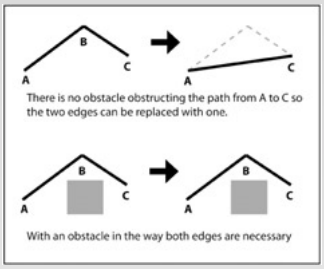
- use A* or Dijkstra depending on whether looking for one or multiple targets

Problem: Kinky Paths



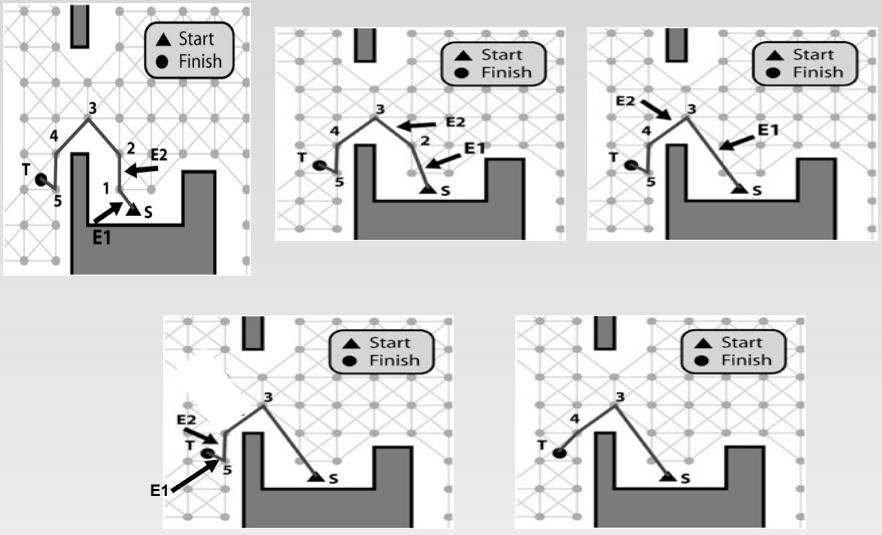
The solution: Path smoothing

Simple Smoothing Algorithm



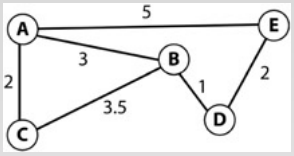
- Check for "passability" between *adjacent* edges

Smoothing Example



DEMO (SMOOTH)

Methods to Reduce CPU Overhead



time/space tradeoff

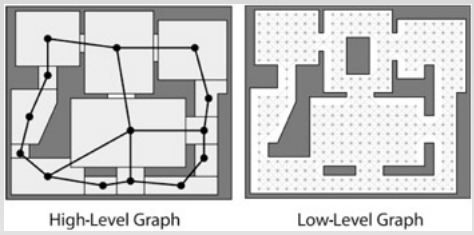
	A	B	C	D	E
A	A	B	C	B	E
B	A	B	C	D	D
C	A	B	C	B	B
D	B	B	B	D	E
E	A	D	D	D	E

shortest path table
(next node)

	A	B	C	D	E
A	0	3	2	4	5
B	3	0	3.5	1	3
C	2	3.5	0	4.5	6.5
D	4	1	4.5	0	2
E	5	3	6.5	2	0

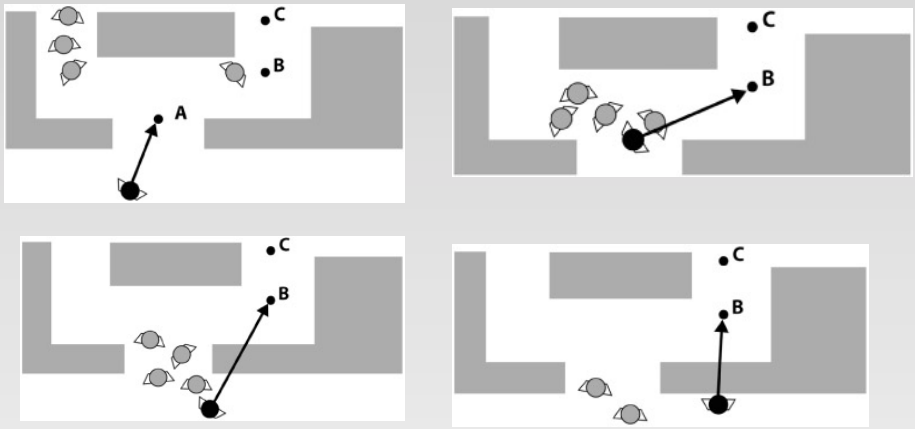
path cost table

Hierarchical Path Planning



- reduces CPU overhead
- typically two levels, but can be more
- first plan in high-level, then refine in low-level

Getting Out of Sticky Situations



- ***bot gets "wedged" against wall***
- ***looks really bad!***

DEMO (STUCK)

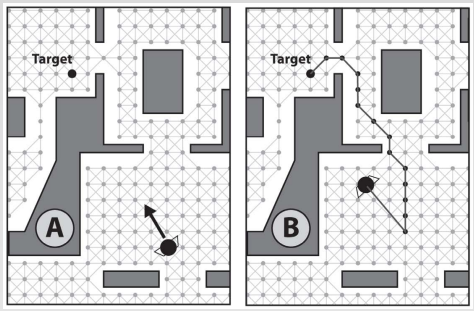
Getting Out of Sticky Situations

- Heuristic:
  - calculate the distance to bot's current waypoint each update step
  - if this value remains about the same or consistently increases
  - then it's probably wedged
  - backup and replan

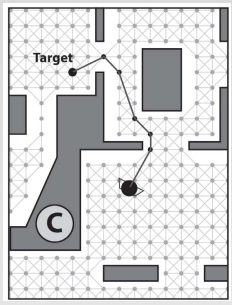
Time Slicing -- Sketch

- When there are many NPC's making calls on the pathfinding module at the same time
- The CPU can get dragged down...
- Solution?
  - evenly divide fixed CPU pathfinding budget between all current callers
  - implies that caller may have to wait for answer
- What should NPC do while it is waiting for path?
  - do not just "block"
  - start moving in "general direction" of target

Time Slicing and Smoothing



without smoothing



smoothed

Advanced Pathfinding Summary

- You would not necessarily use *all* of these techniques in *one* game
- Only use whatever your game demands and no more
- Using *any* of these advanced techniques in final game project counts towards “A”
- For reference C++ code see

http://samples.jbpub.com/9781556220784/Buckland_SourceCode.zip
(Chapter 8 folder)