Path Finding (Navigation)

NPCs usually move, either wandering or aiming for a target (eg the player character)

The space through which they move must be somehow represented

- popular methods are with grids; with waypoint graphs; with navigation meshes

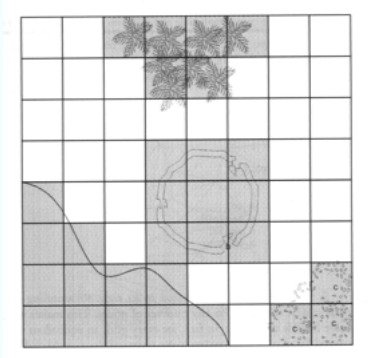
Routes through this space must be found

- For Boss characters you might want nearly optimal search, such as best-first or A*
- For mere creeps you probably want a very flawed form of search
  - random trace
  - breadth-first (probably depth limited)

Grid representations of space for navigation

If using a 2D grid, regularly spaced cells are marked passable or not

- 2½ D world can be handled
  - where there is elevation, but no possibility of two distinct elevations at same position. No bridges crossing paths for example.
- NPC may occupy one or more cells
- One bit per grid cell: little memory needed
  - different grids for different characters?
  - eg for planes, for sharks, for people
- Easy to map a world location to a grid cell



Waypoint representations

Paths that can be safely traversed have their endpoints marked as nodes

- 3D world can be handled just as easily as 2D
- Paths may have diameter associated with them
  - movement need not be centred on path
  - diameter simplifies terrain collision detection
- Paths may have associated traversal cost
  - road vs swamp: effort, time may be measures
    - ie not necessarily distance alone
- Trickier to map world position to waypoint graph
  - Remembering diameter, which path is the red dot on?

- Memory hungry: a node stores paths to neighbours – neighbour, diameter, cost

Navigation Meshes

Those parts of 2.5D space that can be traversed are represented with convex polygons

- Terrain collision detection cheap: it is needed only if an object intersects the straight-line boundaries of a mesh cell
- Traversal is possible
  within area of a polygon,
  or to an adjacent polygon
- Mapping from world
  not as easy as with grids,
  not as hard as with waypoint graphs

- Limited 3D is possible but complicates finding the mesh cell

Path finding algorithms (1)

- Random trace
  - head directly for target if possible, otherwise randomly pick left or right
    - can easily get stuck
    - makes no real attempt to find a *good* path
- Breadth first
  - maintain a queue of partial paths
  - grow the path at head of queue in all possible non-looping ways
  - put the resulting new paths at queue end
  - stop when ***finding*** any path that ends at the goal
    - won't get stuck: it will find a path if there is one
    - the path it finds has fewest links, not necessarily lowest cost

Path finding algorithms (2)

- Best first
  - keep sequence of partial paths, sorted by cost incurred so far.
  - grow cheapest path so far in all possible non-looping ways
  - insert the new paths into the sorted sequence
  - stop when ***about to grow*** a path that has already reached the goal.
    - guarantees to find a path if there is one
    - finds path with lowest cost provided all links have nonnnegative cost
  - can prune search
    - abandoning new paths to an intermediate costlier than another reaching that intermediate

Path finding algorithms (3)

A* search – better because more efficient *if an admissible heuristic is used*

- keep sequence of partial paths, sorted by sum of
  - cost incurred so far
  - and *heuristic lower-bound estimate of cost remaining*.
- grow cheapest-by-sum so far in all non-looping ways and insert into sequence
- can prune search
  - if heuristic's consistency is assured
    - i.e. no link proves cheaper than the reduction in estimated cost
    - admissibility and consistency are different properties
  - or if recording costs of cheapest paths yet to intermediates.

Examples of spaces to be searched

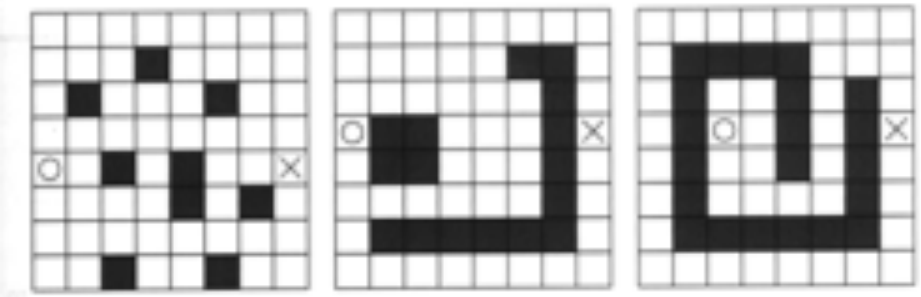
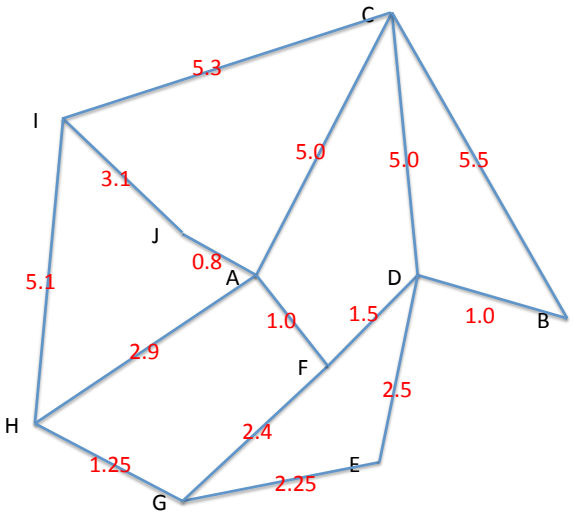


FIGURE 5.4.4 *Three different levels: a) trivial, b) nontrivial, and c) difficult.*

Costs of paths, not used by Breadth-First Search

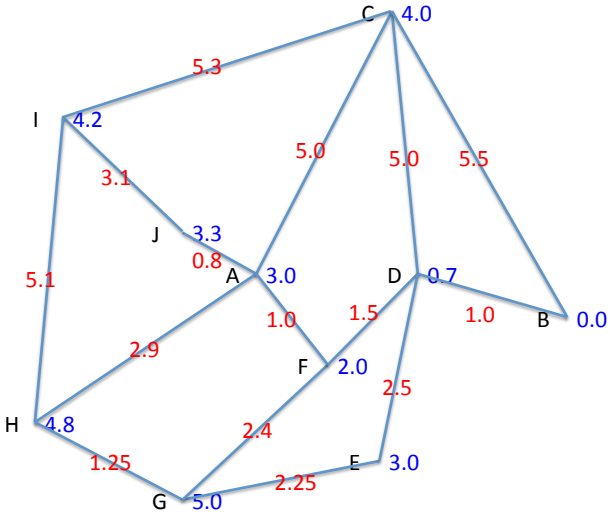


Breadth-first search

A
A-C
A-F
A-H
A-J
A-C-I
A-C-A
A-C-D
A-C-B success

One of the paths with fewest steps is found, A-C-B (there may be others not found with as few steps)
but the cost of those steps may not be minimal

Costs of paths
Estimated Remaining Costs to B, not used by Best-First Search



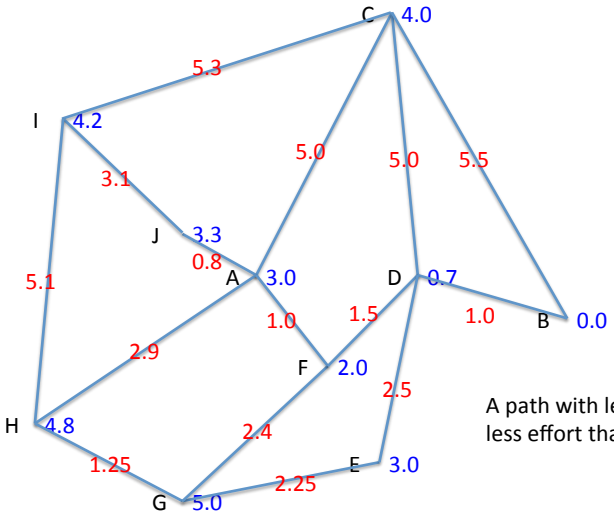
A path with least cost is found, A-F-D-B,
but there is no 'sense of direction'

Costlier Paths can be ignored, e.g. A-H-I costlier than A-J-I

Best-first search

cost	*path*	*added*
0.0	A	1
0.8	A-J	2
1.0	A-F	3
2.5	A-F-D	8
2.9	A-H	4
3.4	A-F-G	10
3.5	A-F-D-B	12
3.65	A-F-G-H	19
3.9	A-J-I	7
4.15	A-H-G	15
5.0	A-C	5
5.0	A-F-D-E	13
5.65	A-F-G-E	20
7.5	A-F-D-C	14
8.0	A-H-I	17
#	A-J-A	6
#	A-F-A	9
#	A-F-D-F	11
#	A-H-A	16
#	A-F-G-F	18

Costs of paths
Estimated Remaining Costs to B



A* search

costs	*path*	*added*
0.0+3.0	A	1
1.0+2.0	A-F	3
2.5+0.7	A-F-D	8
3.5+0.0	A-F-D-B	10
0.8+3.3	A-J	2
2.9+4.8	A-H	4
5.0+3.0	A-F-D-E	11
3.4+5.0	A-F-G	7
5.0+4.0	A-C	5
7.5+4.0	A-F-D-C	12
#	A-F-A	6
#	A-F-D-F	9

A path with least cost is found, A-F-D-B, with less effort than with best-first search

Admissible Heuristic – every node's path cost to goal at least as great as estimate
But Not Consistent –segment G→F's cost is less than the reduction in estimate

Other AI techniques: Flocking, and other emergent behaviours

- When a few simple rules are applied to a host of individuals
- for birds, fish, civilians under fire
  - Separation (steer oneself to avoid collision with others)
  - Alignment (steer in the average direction of others close to self)
  - Cohesion (Steer toward the average position of others close to self)
    - contradictory influences result in realistic behaviour of a flock)
- resulting movement is somewhat arbitrary, undirected
- therefore not very useful for principal characters
- adding additional influences, eg aiming for a particular destination, and/or keeping to one's position in a formation, allows for realistic simulation of the movement of troops

Other AI techniques: Obstacle Avoidance

- Path-finding algorithm (A* or other) may give similar routes for many allied NPCs
- They may easily get in each other's way, especially at choke points
  - crossing narrow bridge
  - running narrow streets
  - exiting cinema

Where "terrain analysis" (coming next) is concerned mainly with static obstacles, "obstacle avoidance" is concerned mainly with moving obstacles

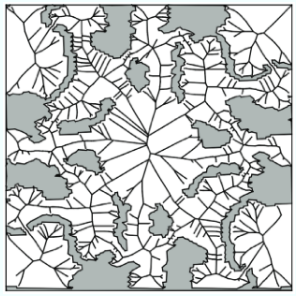
- Combining separation rule with path prediction for allies allows smoother flow

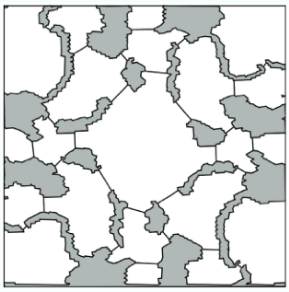
Other AI techniques: Terrain Analysis

https://www.aaai.org/ocs/index.php/AIIDE/AIIDE10/paper/viewFile/2114/2563 , **Moodle**

Apply image-processing techniques to terrain maps (passable-terrain grids) to identify e.g. choke points that impede movements

Perkins's 8-step algorithm





① Recognise obstacle polygons
② Compute Voronoi diagram
③ Prune Voronoi diagram
④ Identify Region nodes
⑤ Identify Choke Point nodes
6. Merge adjacent regions based on thresholds
7. Wall off choke points
8. Recognise region polygons

Steps 1, 2 – Lay groundwork

1. Recognise and simplify polygons

Map data indicate (at least) passable and impassable pixels (including map edge).

Identify polygons – concavities and all – consisting entirely of impassable pixels.

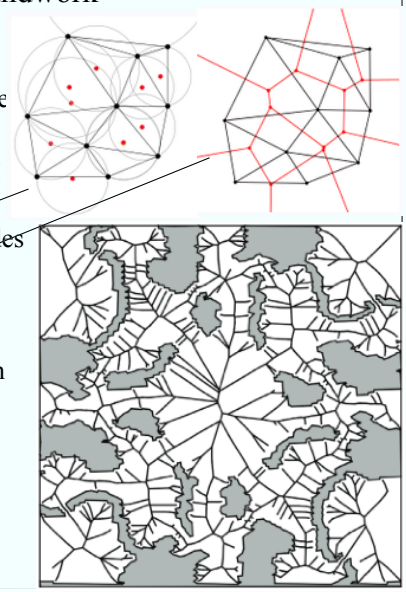
Simplify by eliminating tiny obstacles and the insides of sufficiently-small concave boundaries.

2. Delaunay Triangulation ➔ Voronoi diagram

Delaunay T. of a set of points = a triangulation with no point inside circumcircle of any triangle.

From DT of obstacle promontories, connect the centres of the triangles' circumcircles to form the Voronoi diagram.

Open-source CGAL software exists for this.



Steps 3, 4 – Find significant areas

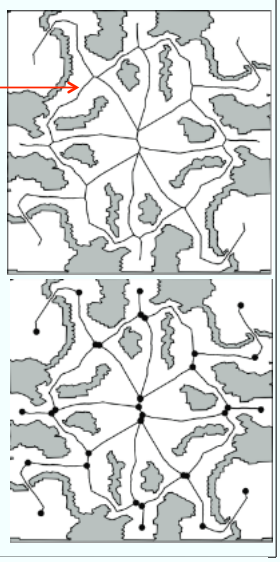
3. Prune Voronoi diagram

Eliminate a leaf node if its circumcircle is smaller than its neighbour's circumcircle.

Since this may result in the neighbour becoming a leaf, iterate while possible.

Then eliminate nodes whose circumcircle is smaller than some threshold.

Note the lines at right often have many segments.



4. Identify Region nodes

Pick out

- nodes with other than two neighbours,
- also nodes with exactly two neighbours when both neighbours' circumcircles are smaller

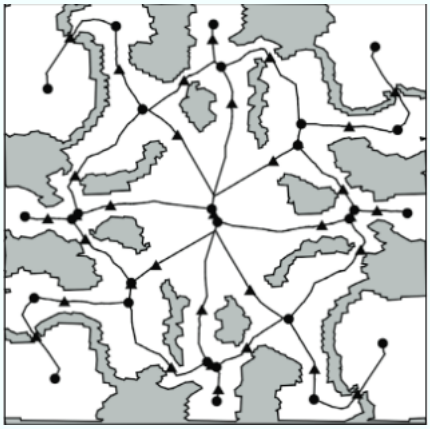
Step 5 – find choke points

5. Find choke points

Nodes that are not region nodes (shown as round) must have exactly two neighbours.

They exist only because they lie on a path between two region nodes.

The node(s) along any path with smallest circumcircle are identified as choke points (shown as triangle)



Steps 6, 7, 8 – Tidy up and simplify

6. Merge adjacent regions

Where the choke point node has a circumcircle nearly as large (85%?) as that of a connected region, eliminate the choke point and merge the connected regions.

7. Wall off choke points

Identify lines passing through a choke point and its adjacent obstacle promontories.

8. Recognise region polygons

Use these walls together with the obstacles to partition the map into regions.

Other AI techniques: Smart Terrain

The technique of putting "intelligence" into inanimate objects

- "affordance theory" – the design of an object suggests the ways it can be used
  - door without handle can be pushed but not pulled
  - microwave oven can cook things
  - food items can satisfy hunger

Player can find out from objects how to use them, what goal states they help achieve

In contrast to affordance theory, functions may be

- present but not suggested (eg trapdoor),
- suggested but not present (eg cardboard chair)