Motivation

- Path-finding
  - A common thing we want to do with NPCs

- But, what is it?
  - Given a start position/state, find a "good" path to a goal position/state
  - Could be a walking/flying path
  - Could be a solution sequence for a puzzle

- Examples
  - Find a path from one place to another, avoiding obstacles
  - Solve an "Eight-Piece" puzzle
A* Algorithm High-Level

- Given:
  - Start state
  - Goal state
  - List of candidate states (nodes): OPEN
  - List of nodes we have tried: CLOSED

- Visit each successor
  - Compute the cost
  - Estimate distance to goal
  - Update cost based on current path
Estimating Cost: $F = G + H$

- We assign to each node:
  - $G$: the movement cost to get from start to here
  - $H$: the estimated cost to get from here to goal
  - $F$: the sum of $G$ and $H$

- We sort OPEN by lower $F$ value:
  - Explore "cheaper" possibilities first

- Choosing a good **heuristic** for $H$ is important
A* Algorithm Pseudocode

1. Create a node containing the goal state node_goal
2. Create a node containing the start state node_start
3. Put node_start on the OPEN list
4. while the OPEN list is not empty {
5.  Get the node off the OPEN list with the lowest f and call it node_cur
6.  if node_cur is the same state as node_goal // We have found the solution!
7.  break from the while loop
8.  Generate each state node_succ that can come after node_cur
9.  for each node_succ of node_cur {
10. Set the cost of node_succ to be the cost of node_cur plus the cost to get to node_succ from node_cur
11. find node_succ on the OPEN list
12. if node_succ is on the OPEN list but the existing one is as good or better
13.  discard this successor and continue // Other path to node_succ is better.
14. if node_succ is on the CLOSED list but the existing one is as good or better
15.  discard this successor and continue // Other path to node_succ is better
16.  Remove occurrences of node_succ from OPEN and CLOSED
17.  Set the parent of node_succ to node_cur
18.  Set h to be the estimated distance to node_goal // Using the heuristic function
19.  Add node_succ to the OPEN list // We'll check this later
20. }
21. Add node_cur to the CLOSED list // We're done processing this node
22. }

R.W. Lindeman - WPI Dept. of Computer Science
Interactive Media & Game Development
A* Algorithm Dissection

- Green: Start
- Red: Goal
- Blue: Barrier

- G: 10 vert/horiz, 14 diag.
- H: Manhattan distance * 10
A* Algorithm (cont.)

- Now check for the low F value in OPEN
  - In this case NE = SE = 54, so choose SE

- Going directly to SE is cheaper than E->SE
  - Leave start as the parent of SE, and iterate
A* Algorithm (cont.)

- Keep iterating until we reach goal, and OPEN is empty
- Follow the parent links to get short path
Choosing a Distance Heuristic (H)

- Any graph-search algorithm is **admissible** if it always returns an optimal solution.
- A* is only admissible if we never overestimate H.
  - H too big: NO guarantee of shortest path, but it will be quick!
  - H = 0: Always gets the optimal path, but will search large space (breadth first)
Examples

- http://www.antimodal.com/astar/

- Now let's do one!
References

- "Steering Behaviors For Autonomous Characters" by Craig Reynolds
  - http://www.red3d.com/cwr/steer/

- "A* Algorithm Tutorial" by Justin Heyes-Jones

- "A* Pathfinding for Beginners" by Patrick Lester