IMGD 3000 - Technical Game Development I: Path-finding AI in Games

by

Robert W. Lindeman

gogo@wpi.edu
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 \texttt{node\_goal}
2. Create a node containing the start state \texttt{node\_start}
3. Put \texttt{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 \texttt{node\_cur}
6. if \texttt{node\_cur} is the same state as \texttt{node\_goal} // We have found the solution!
7. break from the while loop
8. Generate each state \texttt{node\_succ} that can come after \texttt{node\_cur}
9. for each \texttt{node\_succ} of \texttt{node\_cur}  
   
11. Set the cost of \texttt{node\_succ} to be the cost of \texttt{node\_cur} plus the cost to get to \texttt{node\_succ} from \texttt{node\_cur}
12. find \texttt{node\_succ} on the OPEN list
13. if \texttt{node\_succ} is on the OPEN list but the existing one is as good or better
14. discard this successor and continue // Other path to \texttt{node\_succ} is better.
15. if \texttt{node\_succ} is on the CLOSED list but the existing one is as good or better
16. discard this successor and continue // Other path to \texttt{node\_succ} is better
17. Remove occurrences of \texttt{node\_succ} from OPEN and CLOSED
18. Set the parent of \texttt{node\_succ} to \texttt{node\_cur}
19. Set \(h\) to be the estimated distance to \texttt{node\_goal} // Using the heuristic function
20. Add \texttt{node\_succ} to the OPEN list // We'll check this later
21. }
22. Add \texttt{node\_cur} to the CLOSED list // We're done processing this node
23. }
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