Lane Pollock

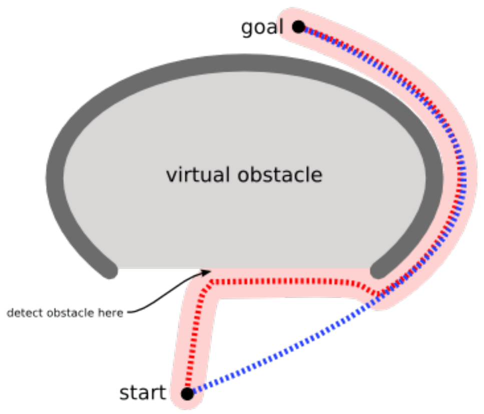
Dr. Roark

May 2, 2025

A* Search Algorithm

The A* Search Algorithm is a complex, greedy path-finding algorithm that will find a good path to another spot when there may be obstacles on the map. It's similar to Dijkstra's shortest path as it can certainly choose the shortest path in the right circumstance, but also uses heuristics, keeping efficiency up. Because it considers both the closest vertices to the starting point, like Dijkstra's, AND vertices closest to the goal, like greedy best-first-searches, it is able to map an ideal path around things. Therefore, the A* algorithm is used popularly in game navigation, to find the best(or smartest seeming) path to a destination around obstacles.

The A* Search Algorithm was created by Peter Hart, Nils Nilsson, and Bertram Raphael in 1968. They sought to create a system for a robot named Shaky that could allow it to determine it's own path whilst there were obstacles around. Imagine the start point and a destination with a wall separating them. Instead of going straight to the wall, then following the length before going around, the algorithm will simply choose to go around the wall from the start. Together, they combined pieces of the two aforementioned algorithm and developed the A* search algorithm.



The algorithm works by considering both the *exact* cost from the starting point to another point on the graph, called g(n), and the *estimated* cost from vertex n to goal, called h(n). The A* balances the two as it moves towards the goal. Each time in the main loop, it checks which vertex has the lowest f(n) = g(n) + h(n). To calculate these distances, several different methods can be used. One is the Manhattan Distance, which is the sum of the of the absolute values of differences in the goal's x and y coordinates, and current cell's x and y coordinates, respectively. That's used when we can only move in 4 directions. Another is the Diagonal distance, which uses the absolute values of the differences in the goal's x and y coordinates and the cell's, respectively. Those are used in the equation This is for when we can move 8 way like in chess. Below is an example from geeksforgeeks.org:

dx = abs(current_cell.x – goal.x)

dy = abs(current_cell.y – goal.y)

h = D * (dx + dy) + (D2 - 2 * D) * min(dx, dy)

where D is length of each node(usually = 1) and D2 is diagonal distance between each node (usually = sqrt(2)).

The third is the Euclidian Distance which uses the distance formula to calculate the pure distance from the goal. The Time complexity worst case is O(E) where E is the edges in the graph. For auxiliary space, it is O(V) where V is the number of vertices.

This algorithm is used in games popularly, because it is one of the best around. It is excellent when you have one source and one destination, especially if it may be impeded in some way or propose a movement hinderance (like only moving 4-way or 8-way). In Unreal Engine for game dev, there is a property called a NavMesh, which you can use to highlight the ground area which an AI can move. This visually shows how it deems certain things like rocks and walls as obstacles that cannot be stepped on, and it even looks like a grid. I can image an algorithm like A* will handle the path the AI will take to a destination based on the graph that was created by the NavMesh.

The Search function itself is quite long and takes up roughly 15 or more pages on a document like this with the spacing present, so below are some implementations on calculating h and tracing a path from the source to destination.

```
double calculateHValue(int row, int col, Pair dest)
{
    // Return using the distance formula
    return ((double)sqrt(
        (row - dest.first) * (row - dest.first)
        + (col - dest.second) * (col - dest.second)));
}

// A Utility Function to trace the path from the source
// to destination
void tracePath(cell cellDetails[][COL], Pair dest)
{
    printf("\nThe Path is ");
    int row = dest.first;
    int col = dest.second;

    stack<Pair> Path;

    while (!(cellDetails[row][col].parent_i == row
             && cellDetails[row][col].parent_j == col)) {
        Path.push(make_pair(row, col));
        int temp_row = cellDetails[row][col].parent_i;
        int temp_col = cellDetails[row][col].parent_j;
        row = temp_row;
        col = temp_col;
    }
    Path.push(make_pair(row, col));
    while (!Path.empty()) {
        pair<int, int> p = Path.top();
        Path.pop();
        printf("-> (%d,%d) ", p.first, p.second);
    }

    return
}
```

The A* Search Algorithm is a super cool algorithm that solves a pretty complex problem in path finding. Even better, it does so efficiently thanks to its use of heuristics alongside the power of Dijkstra's algorithm. The balance between those is what makes the A* so nice for use in video games like Tower Defense. Because of it's use of heuristics, the A* may not find the best path every time, but will provide great efficiency when there's one start and one destination.

Sources:

Kumar, Rajesh. "The A* Algorithm: A Complete Guide." *DataCamp*, DataCamp, 7 Nov. 2024, www.datacamp.com/tutorial/a-star-algorithm.

"A* Search Algorithm." *GeeksforGeeks*, GeeksforGeeks, 30 July 2024, www.geeksforgeeks.org/a-search-algorithm/.

*Introduction to A**, theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html. Accessed 2 May 2025.