AStar Algorithm Explained

AStar Algorithm

What Is It?

In Unity 3D paths can be calculated using Nav Meshes and Nav Mesh Agents.

In Unity 2D there are no built-in solutions to pathfinding – so we have to create our own.

The AStar algorithm is used in pathfinding and is a good solution for 2D pathfinding.

It can efficiently find the shortest route between two points in a ‘graph’ of connected nodes.

In our 2D game the ‘graph’ is a tilemap grid, which is well suited to the AStar algorithm.

AStar Algorithm

How Does It Work?

Each grid square, or node, has a 'total cost' based on the distance travelled on an evaluated path from the start node to the finish node.

Every node that has been 'evaluated' will have a path back to the start node (this path is maintained by recording the parent for every node – so that the path back to the start can be traced back through the parents).

The 'GCost' for a node is the sum of distances on this path back to the start node. The 'GCost' for nodes can 'improve' if shorter paths to the node are found by the algorithm.

The 'HCost' for a node is a simple calculation of the minimum distance to the finish. For example if the horizontal distance between squares is 10 units and the finish square is 3 horizontal squares away, the HCost would be 30. Note that this is the minimum distance and may not be traversable if there is an obstacle in the way. The HCost doesn't change for a node.

The 'total cost' for a node, known as the 'FCost', is the sum of the GCost and the HCost. The shortest path is determined by evaluating the nodes and selecting the path with the lowest FCost from the start to the finish.

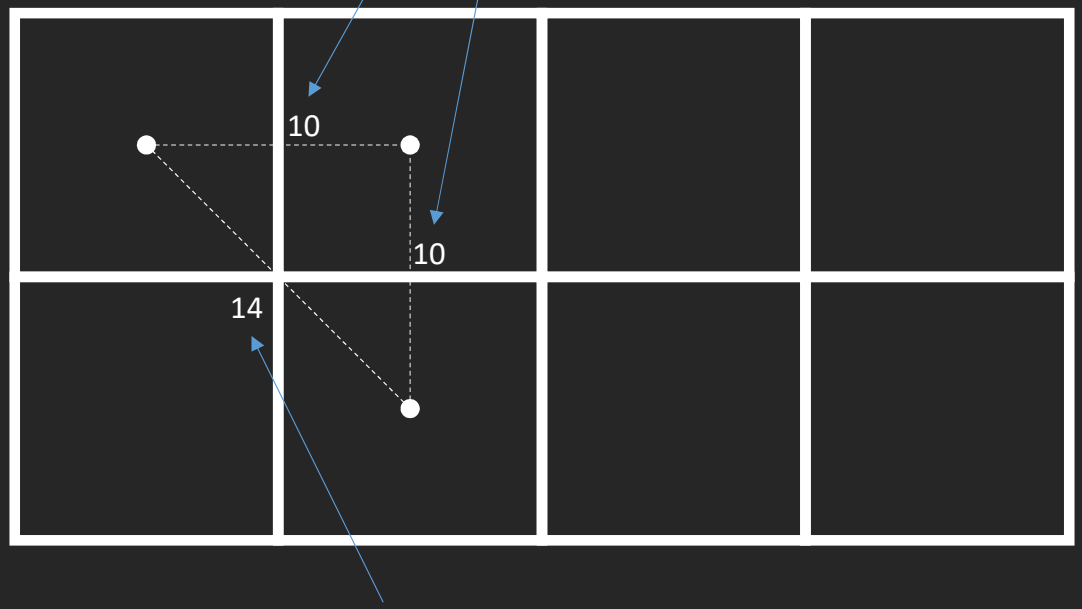
The algorithm maintains 'open' and 'closed' lists. From the start square it adds all surrounding squares to the open list. It selects the square in the open list with the lowest FCost, adds it to the closed list, and then repeats this evaluation for all its neighbours. This continues until the target node is reached, and the shortest path is found.

AStar Algorithm

Distance Between Square Nodes

In a grid the distance is either the horizontal distance between squares or the diagonal distance between squares.

Distance Between 2 Squares Horizontally Or Vertically is 10 units



Using Pythagoras Distance Between 2 Squares Diagonally = $\sqrt{10^2 + 10^2}$ = approximately 14

AStar Algorithm

Pseudo Code Logic

Open List = list of nodes/squares to be evaluated.
Closed List = list of nodes/squares already evaluated.
Start = start node.
Target = target finish node.

Add Start to the Open List.
WHILE there are nodes in Open List
- Retrieve the Current Node with the lowest FCost (use lowest HCost as tie breaker)
- IF Current Node = Target then path found and finish.
- Move the Current Node from the Open List to the Closed List.
- LOOP through each of the 8 neighbour nodes.
    - IF the neighbour node is an obstacle or in the Closed List then skip it.
    - Calculate values for the neighbour node:-
        - GCost from the Start based on the path to the Current Node.
        - HCost to the Target based on the minimum distance.
        - FCost = GCost + HCost
    - IF the neighbour node is not in the Open List OR it is in the Open List but it's recalculated FCost is lower
        - Save the GCost, HCost & FCost for the neighbour node.
        - Set the parent of the neighbour node to the Current Node.
        - IF the neighbour node isn't in the Open List then add it.

AStar Algorithm

Lists and HashSets

For the Open List we are going to use the C# List<T> generic collection. The 'T' will be the grid square node. This is because we want to sort the list of nodes to easily retrieve the node with the lowest FCost.

For the **Closed List** we are going to use the C# HashSet<T> generic collection. The 'T' will be the grid square node. We don't need to sort the Closed List – only check whether a node is already in the closed list. The 'contains' method on HashSets does this and is very fast since it doesn't have to iterate through each item - it can check directly using the hash value it's calculated.