

Yazan Mahmoud Barakat

411010698

CSC321

A Star Algorithm

**A\***

**Introduction About Path Finding**

**Problem To Solve: Path Finding**

The idea of path planning does exist in a lot of different fields such as AI in games, robots, and even maps, In the world of game development and while making an AI for a game you will face a famous challenge and It Is how is your AI agent going to move inside your game, There Is a Lot of Technics and Your Solution to Choose May Be Different Depending on The Type of Game You Are Making.

Some Game Engines Have Built-in Functionality to Help You with Your AI But in Our Case, We Are Going to Cover One of The Best Algorithms Which is A Star (A\*)

**What Is A\***  
A\* Is an Algorithm that is mostly used in pathfinding applications and it is one of the most used algorithms for AI inside video games, it is known to be one of the best algorithms for path planning and it uses a combination of heuristic searching and searching based on the shortest path

**What Is the Time Complexity Of A\***

**A star** is Dijkstra with a heuristic that fulfills some properties You can select different heuristic functions that lead to different time complexities. The simplest heuristic is **straight-line distance**. However, there is also more advanced stuff like **landmarks heuristic** for example one of the functions can be the next :

F(v) = h(v) + g (v)

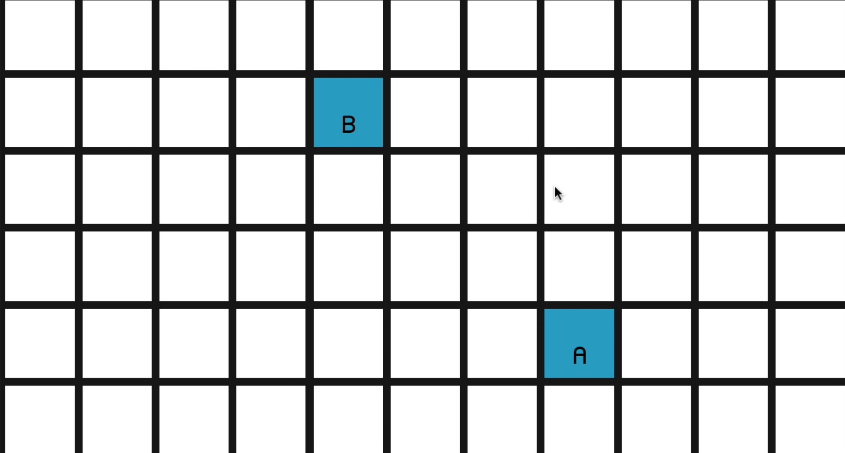
Where h(v) is a heuristic distance of the cell to the goal state, and g(v) is the length of the path from the initial state to the goal state through the selected sequence of cells

**Advantages vs Disadvantages of A\***

|  |  |
| --- | --- |
| **Advantages** | **Disadvantages** |
| It is complete and optimal. | This algorithm is complete if the branching factor is finite and every action has a fixed cost. |
| It is the best one from other techniques | The speed execution of the A\* search is highly dependent on the accuracy of the heuristic algorithm that is used to compute h(n). |
| It is optimally efficient, i.e., there is no other optimal algorithm guaranteed to expand fewer nodes than A\*. | It has complexity problems. |

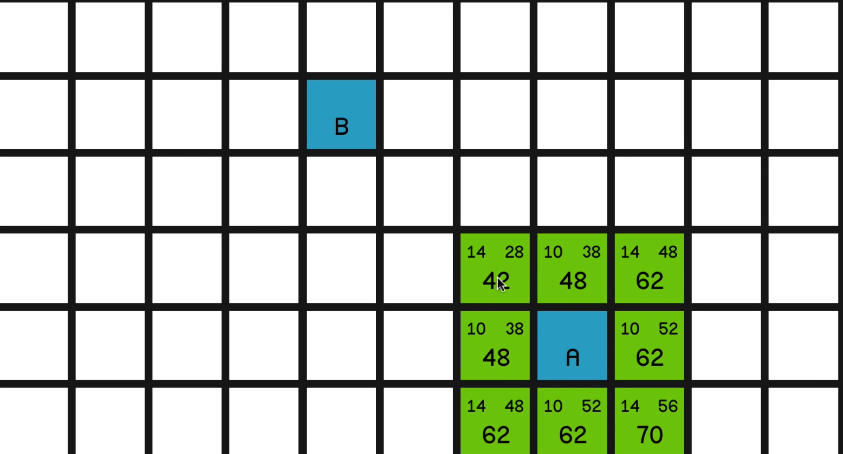
**How A\* Works**

First of all, we need to cut our world space to a grid-based world so we can calculate each cell value and assign it,  
In our grid system, we need to have a starting point (A) and a target point (B),  
and We want to find the closest path between points A and B

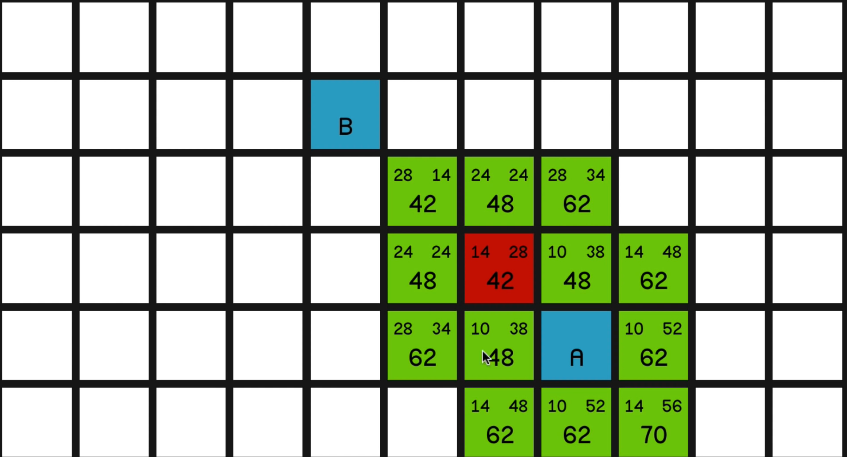


After placing your points the algorithm will start calculating 3 values for each cell that is neighbor to the cell, we have in our open list that is a neighbor to a cell in our closed list (The starting point should be in the closed list from the beginning)  
The values we will calculate are:

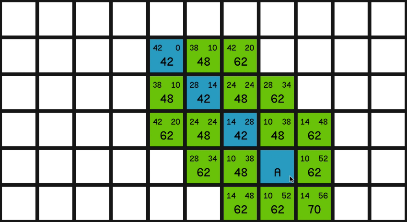
1. G Cost: Is: How Far This Cell Is from The Starting Point
2. H Cost: How Far This Cell Is from The End Point
3. F Cost: Is G Cost + H Cost



The Algorithm Will Chose the Cell with The Lowest F Cost and Will Mark It as Closed (Part of The Path) After That It Will Calculate All the Values Again for The New Cell We Have



In the end, we will have our path



Here Is Another Example If We Had an Obstacle



**Alternative algorithms**

There Are Many Different Algorithms for Path Finding Each One with Its Pros and Cons for Example

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Algorithm | Will Always Return the Shortest Path | Support Movement Cost | Support Multiple Start / End Point | Time Complexity | Visualization |
| Breadth Frist Search | Yes | No | Yes | O(V^2) |  |
| Dijkstra | Yes | Yes | Yes | O(V^2) |  |
| A\* | Hard To Decide (heuristic) | Yes | No | (Depends on the implementation)  O(b\*d) |  |

**A\* Algorithm Pseudocode Code**

Open List to Store the Nodes That Is Not evaluated yet

Closed List to Store the Nodes That have been evaluated

Loop

Current Node in open with the lowest f cost

Remove current from Open

Add current to closed

If(current is the end node)

Return path found

Foreach neighbor of the current node

If the neighbor is not traversable or neighbor is in closed

Skip to the next neighbor

If the new path to the neighbor is shorter or the neighbor is not in open

Set f cost of the neighbor

Set parent of neighbor to current

If the neighbor is not in open

Add neighbor to open

To Apply This Algorithm, we would need two more Classes, the first one to identify what is a node in the world and its properties what is the f cost, g cost, are allowed to walk here or not so We Created the class Node

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Node

{

public int gridX, gridY;

public int gCost, hCost;

public int fCost{get { return gCost + hCost;

}}

public Node parent;

public bool walkable;

public Vector3 worldNodePos;

public Node(bool \_walkable , Vector3 \_worldNodePos , int \_gridX , int \_gridY)

{

gridX = \_gridX ;

gridY = \_gridY;

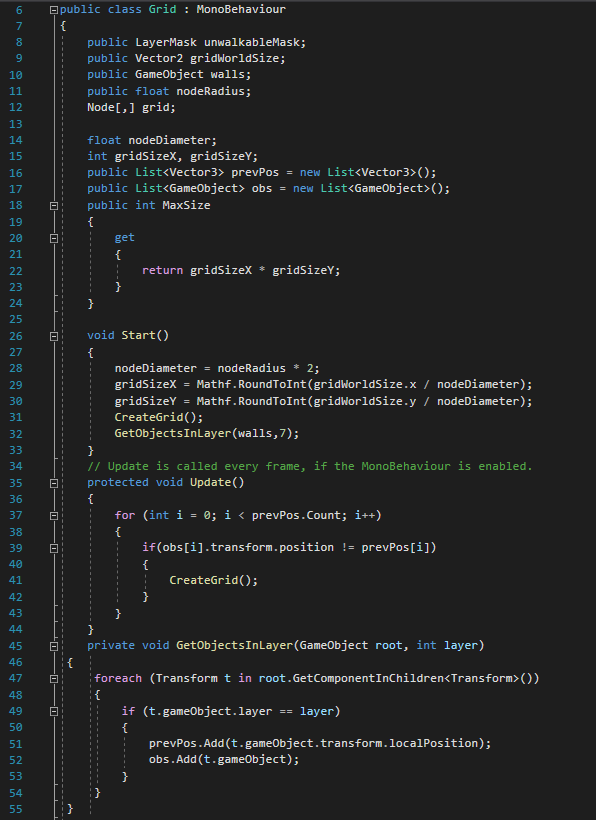
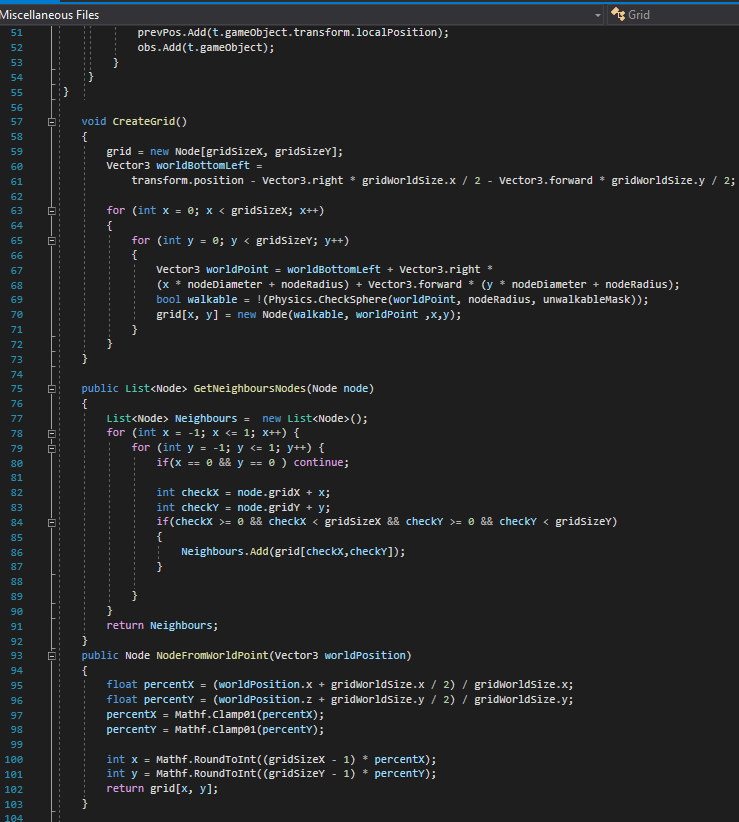
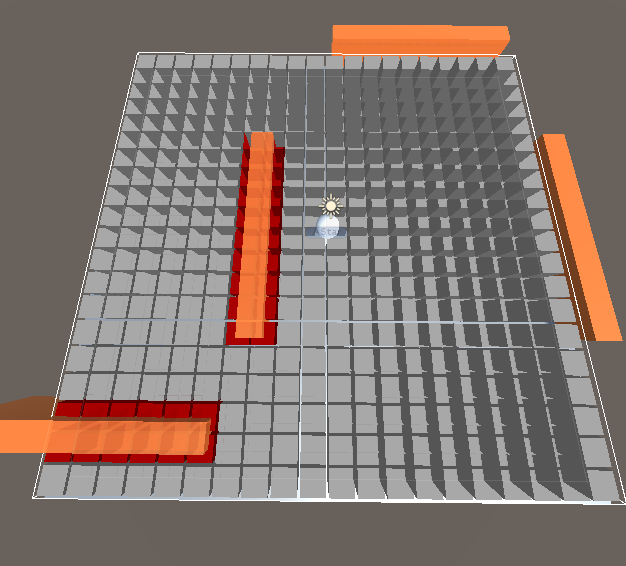
walkable = \_walkable;

worldNodePos = \_worldNodePos;

}

}

And After That, We Need a Class to Give the Algorithm the world size and it will cut into a grid and check for each node’s stats so We Created the Grid Class

 **Applied Code in C#**

using System;

using System.Collections.Generic;

using UnityEngine;

using System.Diagnostics;

public class AStarPathFinding : MonoBehaviour

{

Grid grid;

public Transform player;

public Transform target;

protected void Awake()

{

grid = GetComponent<Grid>();

}

void FindPath(Vector3 startPos, Vector3 targetPos)

{

Stopwatch sw = new Stopwatch();

sw.Start();

Heap<Node> openSet = new Heap<Node>(grid.MaxSize);

HashSet<Node> closedSet = new HashSet<Node>();

Node startNode = grid.NodeFromWorldPoint(startPos);

openSet.Add(startNode);

Node endNode = grid.NodeFromWorldPoint(targetPos);

while (openSet.Count > 0)

{

Node currentNode = openSet.RemoveFirst();

closedSet.Add(currentNode);

if (currentNode == endNode)

{

print($"Path Found In {sw.ElapsedTicks} ms");

RetractPath(startNode, endNode);

return;

}

foreach (Node neighbour in grid.GetNeighboursNodes(currentNode))

{

if (!neighbour.walkable || closedSet.Contains(neighbour))

{

continue;

}

int newMovementCostToNeighbour = currentNode.gCost + GetDistance(currentNode, neighbour);

if (newMovementCostToNeighbour < neighbour.gCost || !openSet.Contains(neighbour))

{

neighbour.gCost = newMovementCostToNeighbour;

neighbour.hCost = GetDistance(neighbour, endNode);

neighbour.parent = currentNode;

if (!openSet.Contains(neighbour))

openSet.Add(neighbour);

else

{

//openSet.UpdateItem(neighbour);

}

}

}

}

}

int GetDistance(Node a, Node b)

{

int dstX = Mathf.Abs(a.gridX - b.gridX);

int dstY = Mathf.Abs(a.gridY - b.gridY);

if(dstX > dstY)

{

return 14 \* dstY + 10\*(dstX - dstY);

}

return 14\* dstX + 10\*(dstY - dstX);

}

void RetractPath(Node startNode , Node endNode)

{

List<Node> path = new List<Node>();

Node currentNode = endNode;

while (currentNode != startNode)

{

path.Add(currentNode);

currentNode= currentNode.parent;

}

path.Reverse();

grid.path = path;

}

// Update is called once per frame

void Update()

{

if (Input.GetKeyDown(KeyCode.Space))

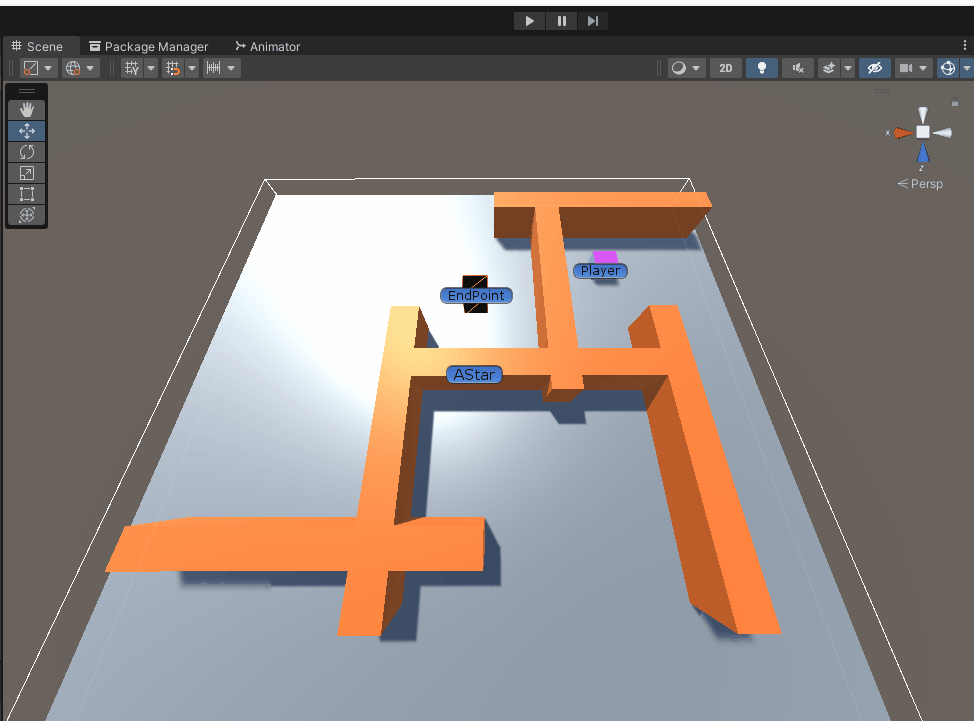
{

FindPath(player.position, target.position);

}

}

}



**Resources**

1. Sebastian Lague
2. GameDev.TV
3. <https://qiao.github.io/PathFinding.js/visual/>
4. [www.ccpo.odu.edu](http://www.ccpo.odu.edu)
5. [www.scaler.com](http://www.scaler.com)
6. <https://www.geeksforgeeks.org>
7. <https://www.sciencedirect.com/science/article/pii/S187770581403149X>
8. <https://www.vtupulse.com/artificial-intelligence/a-star-search-algorithm-artificial-intelligence/>
9. <https://cs.stackexchange.com/questions/56176/a-graph-search-time-complexity>