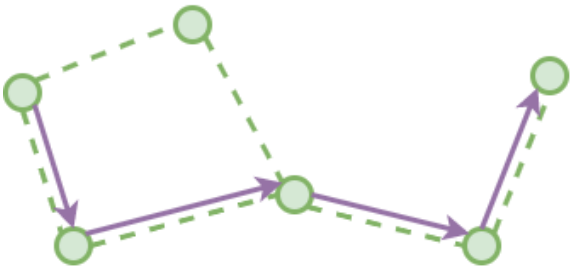
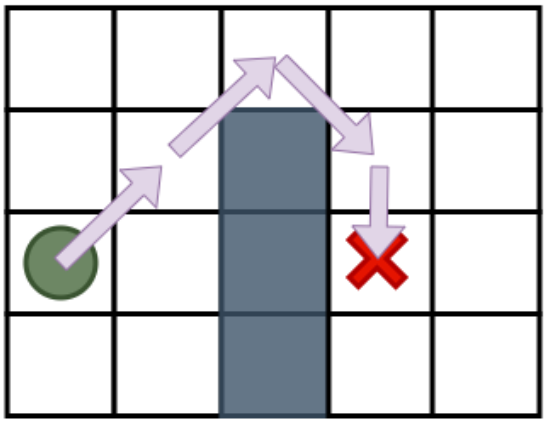
Aoiti's Ultimate A* Pathfinding Solution

This A* pathfinding class is designed to fit any situations as long as the class is supplied with a method that calculates a heuristic approximation of distance between two nodes/points or any other objects, and a method that returns connected nodes/points or objects of same type and their distances as floats.

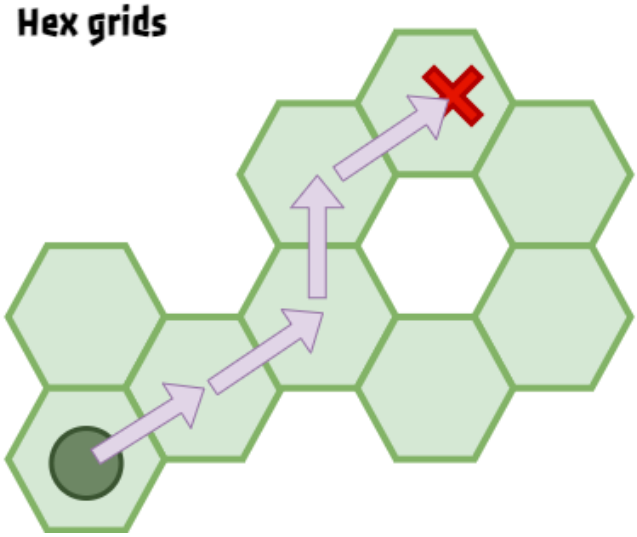
Applicable to Nodes,



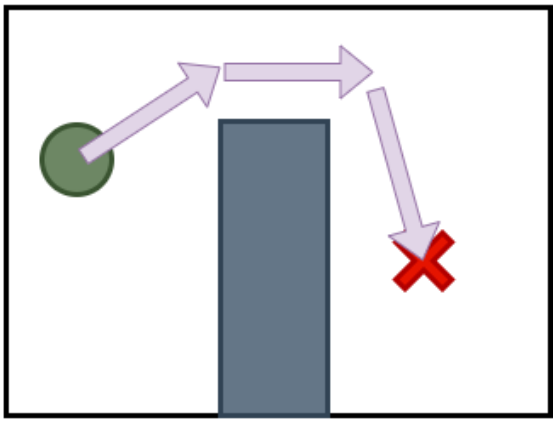
Square tiles



Hex grids



Scenes with colliders



Define the following

T : Any class or struct that has a way to measure distance between its instances. Examples are; Vector3, Vector2, Vector3Int etc.

float HeuristicDistance(T pointA, T pointB) : any method name that takes two T instances and returns an aproximate distance

Dictionary<T, float> ConnectedNodesAndStepCosts(T centerPoint) : any method name that takes a T instance and returns a dictionary of its neighbours as the keys and distances to the neighbours as the values.

int calculatorPatience: the number of steps that the algorithm calculate before giving up. It is a safety precaution for testing. I advise starting with 100 or less and increasing to 9999 or more after testing.

Usage

using **Aoiti.Pathfinding;**
...
Pathfinder<T>() mypathfinder= **new Pathfinder**<Vector3>(**HeuristicDistance**, **ConnectedNodesAndStepCosts, calculatorPatience**); //Call only once
...
List<**T**> **path**;
mypathfinder.**GenerateAstarPath(A, B,** out **path);** //A and B are instances of T

Explanation

using Aoiti.Pathfinding : package name

Pathfinder<T>(HeuristicDistance, ConnectedNodesAndStepCosts, [calculatorPatience=9999]) : creates a pathfinder solution for your particular need.

bool GenerateAstarPath(T startPoint, T endPoint, out List<T> path) : returns true if exists a path from starting instance to ending instance, and outputs an array of adjacently connected T instances that lead to the target (**path**).

Pathfinder structure

```
1.  namespace Aoiti.Pathfinding
2.  {
3.      public class Pathfinder<T>
4.      {
5.          public Pathfinder(Func<T, T, float> HeuristicDistance, Func<T, Dictionary<T,
    float>> ConnectedNodesAndStepCosts, int calculatorPatience = 9999);
6.
7.          public bool GenerateAstarPath(T startNode, T targetNode, out List<T> path);
8.      }
9.  }
```

Example

Use of pathfinder can be as simple as below in a scene with colliders to use a 3D grid to find path;

NavigationTest.cs

```
1.  using System.Collections.Generic;
2.  using UnityEngine;
3.  using Aoiti.Pathfinding; //import the pathfinding library
4.
5.
6.  public class NavigationTest: MonoBehaviour
7.  {
8.      Pathfinder<Vector3> pathfinder;
9.      List<Vector3> path = new List<Vector3>();
10.
11.     private void Start()
12.     {
13.         pathfinder = new Pathfinder<Vector3>(GetDistance, GetNeighbourNodes);
14.     }
15.
16.     private void Update()
17.     {
18.         if (Input.GetMouseButtonDown(0)) //check for a new target
19.         {
20.             Vector3 target = Camera.main.ScreenToWorldPoint(Input.mousePosition);
21.             if (pathfinder.GenerateAstarPath(transform.position, target, out path))
    //if there is a path from current position to target position reassign path.
22.         }
23.
24.         transform.position = path[0]; //go to next node
25.         path.RemoveAt(0); //remove the node from path
26.
27.     }
28.
29.     float GetDistance(Vector3 A, Vector3 B)
30.     {
31.         return (A - B).sqrMagnitude;
32.     }
33.
34.     Dictionary<Vector3, float> GetNeighbourNodes(Vector3 pos)
35.     {
36.         Dictionary<Vector3, float> neighbours = new Dictionary<Vector3, float>();
37.         for (int i = -1; i < 2; i++)
38.         {
39.             for (int j = -1; j < 2; j++)
40.             {
41.                 for (int k=-1;k<2;k++)
42.                 {
43.
44.                     if (i == 0 && j == 0 && k==0) continue;
45.
46.                     Vector3 dir = new Vector3(i, j,k);
47.                     if (!Physics.Linecast(pos, pos + dir))
48.                     {
49.                         neighbours.Add(pos + dir, dir.magnitude);
50.                     }
51.                 }
52.             }
53.
54.         }
```

```
55.         return neighbours;
56.     }
57.
58. }
```