#CG/code #NTCC/Iteration-6 \_\_\_

#### Dissection:

using UnityEngine;  
using System.Collections;  
using System.Collections.Generic;  
  
public class PathFinding : MonoBehaviour  
{  
 private Grid \_grid; // Grid refrence  
  
 public void FindPath(Node startPos,Node endPos, Grid grid) // start node and target node are passed  
 {  
 Node startNode = startPos;  
 Node targetNode = endPos;  
 \_grid = grid;  
  
 List<Node> openSet = new List<Node>(); // list for the node yet to be calculated  
 HashSet<Node> closedSet = new HashSet<Node>(); // list of node that are calculated  
 openSet.Add(startNode);

while (openSet.Count > 0)  
 {  
 Node node = openSet[0];  
 for (int i = 1; i < openSet.Count; i++)  
 {  
 if (openSet[i].fCost < node.fCost || openSet[i].fCost == node.fCost)  
 {  
 if (openSet[i].hCost < node.hCost)  
 node = openSet[i];  
 }  
 }

openSet.Remove(node);  
 closedSet.Add(node);  
  
 if (node == targetNode)  
 {  
 RetracePath(startNode, targetNode);  
 return;  
 }

foreach (Node neighbour in grid.GetNeighbours(node))  
 {  
 if (!neighbour.walkable || closedSet.Contains(neighbour))  
 {  
 continue;  
 }

int newCostToNeighbour = node.gCost + GetDistance(node, neighbour) + node.getMovePenalty();  
 if (newCostToNeighbour < neighbour.gCost || !openSet.Contains(neighbour))  
 {  
 neighbour.gCost = newCostToNeighbour;  
 neighbour.hCost = GetDistance(neighbour, targetNode);  
 neighbour.parent = node;  
  
 if (!openSet.Contains(neighbour))  
 openSet.Add(neighbour);  
 }  
 }  
 }  
 }

void RetracePath(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;  
  
 }

int GetDistance(Node nodeA, Node nodeB)  
 {  
 int dstX = Mathf.Abs(nodeA.gridX - nodeB.gridX);  
 int dstY = Mathf.Abs(nodeA.gridY - nodeB.gridY);  
  
 if (dstX > dstY)  
 return 14 \* dstY + 10 \* (dstX - dstY);  
 return 14 \* dstX + 10 \* (dstY - dstX);  
 }  
}