CT3536
(Games Programming using Unity3D)

Section 10

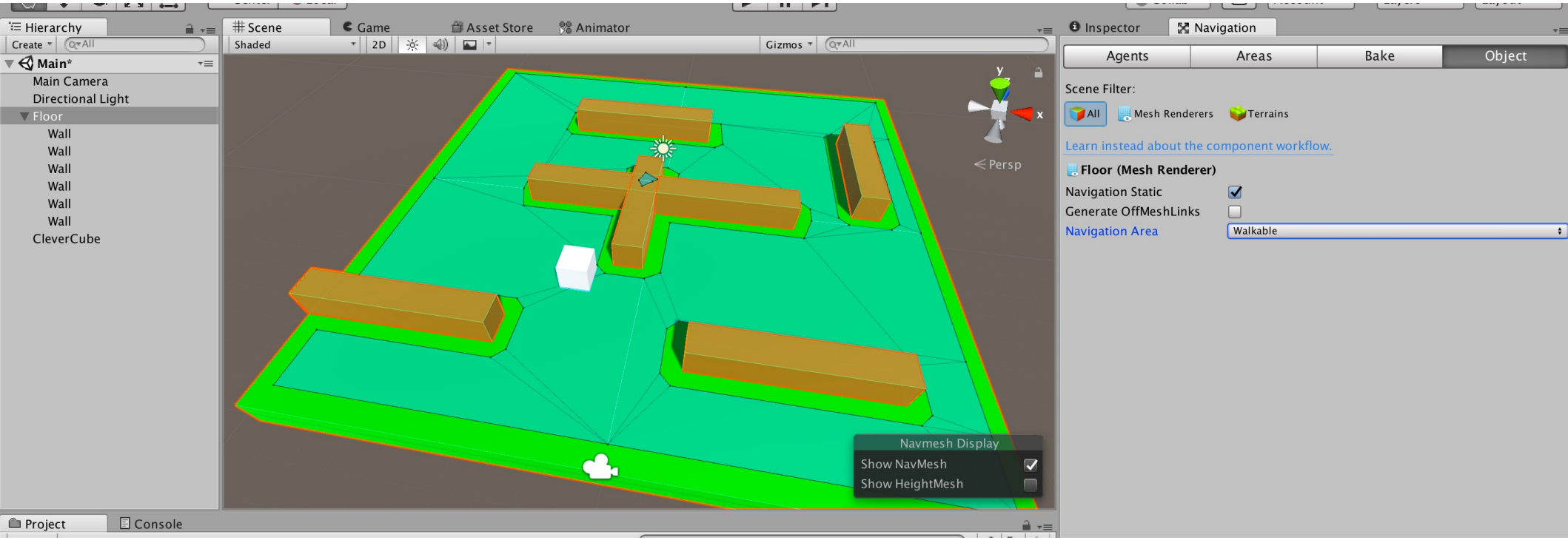
Pathfinding
C# Threads using “Thread Ninja”

Pathfinding in Unity using NavMesh and NavMeshAgent

- See demo project on Canvas “NavMeshDemo”
- This is loosely based on:

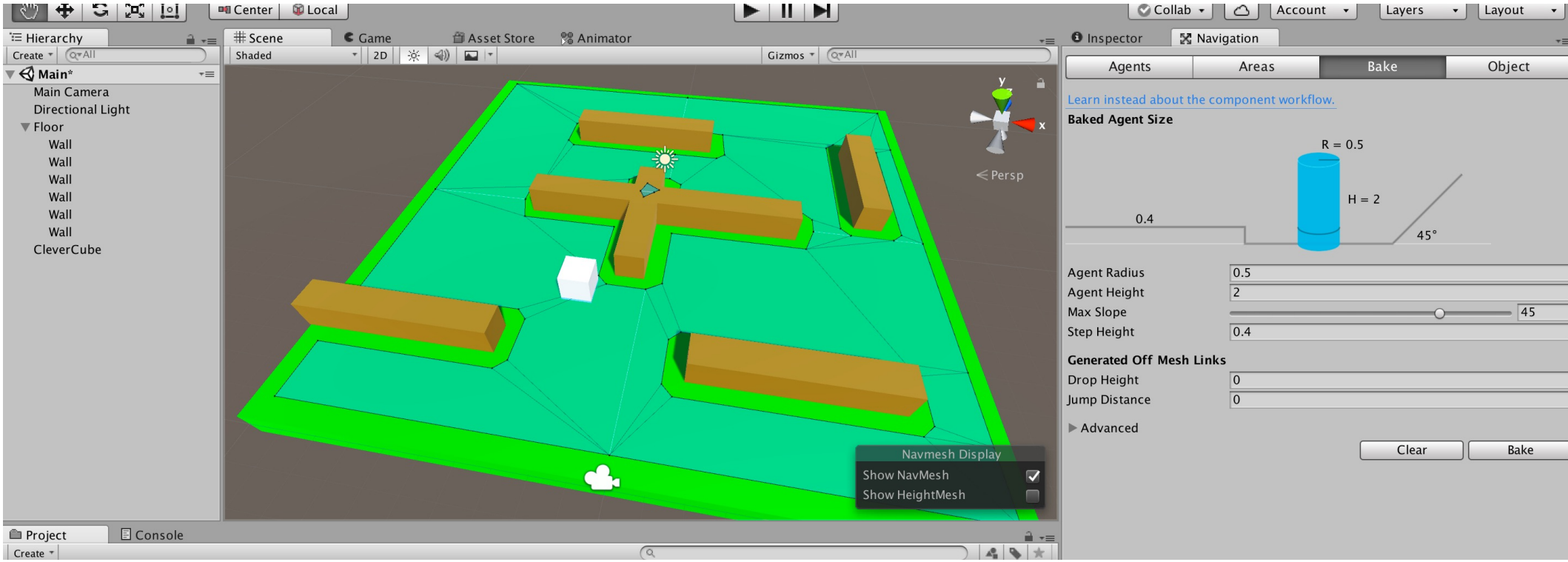
https://www.red-gate.com/simple-talk/dotnet/c-programming/pathfinding-unity-c/

- Use Window>Navigation to open the Navigation tab.
- Here we’re creating a NavMesh attached to Floor

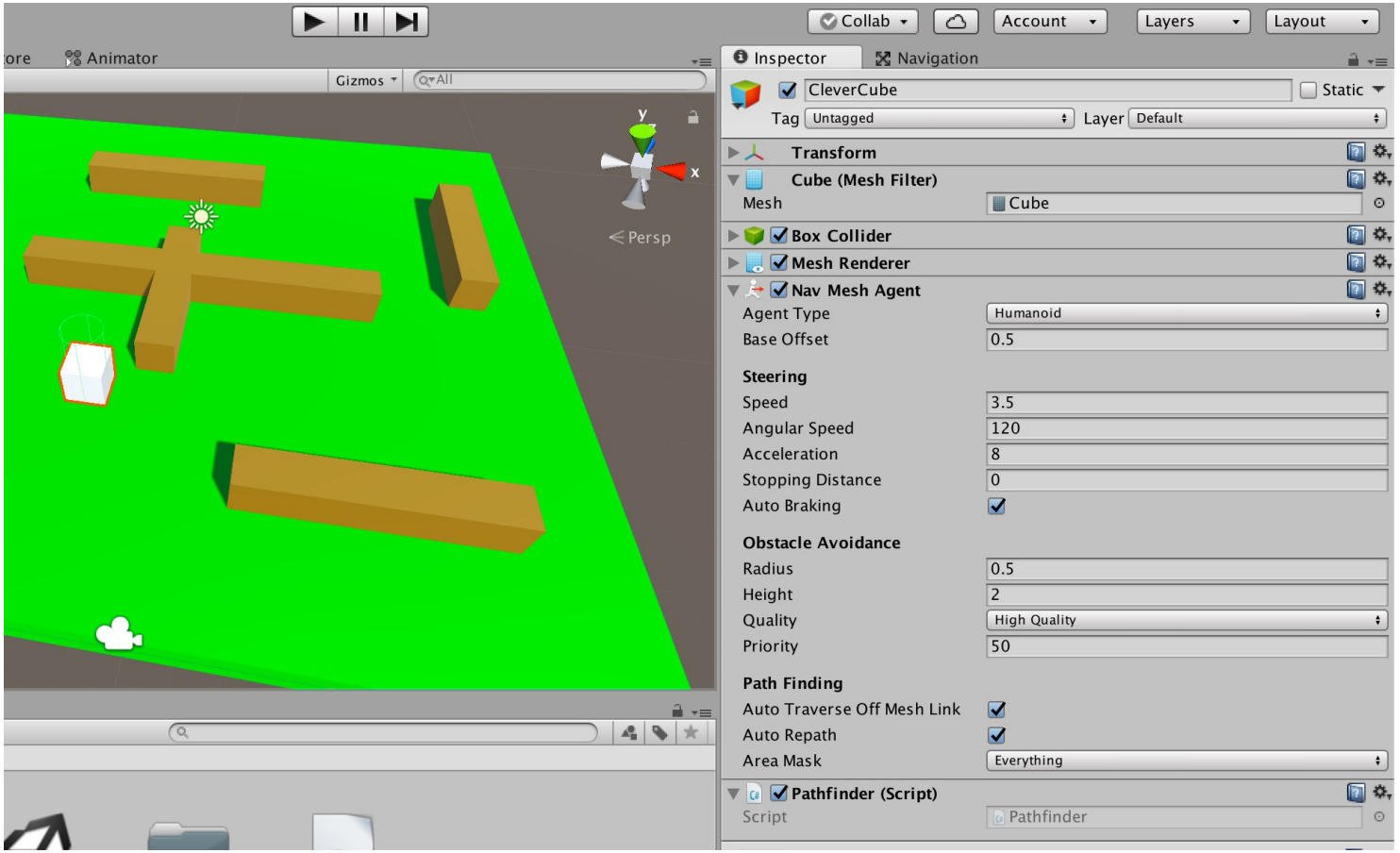


Pathfinding in Unity using NavMesh and NavMeshAgent

- In the ‘Bake’ settings you have various options
- The ‘Bake’ button creates the mesh
- This sparse navmesh has far fewer nodes than the grid-based approach we take below => more efficient to use



The CleverCube Object



- Has a NavMeshAgent component added
- Has a new custom script added: ‘Pathfinder’

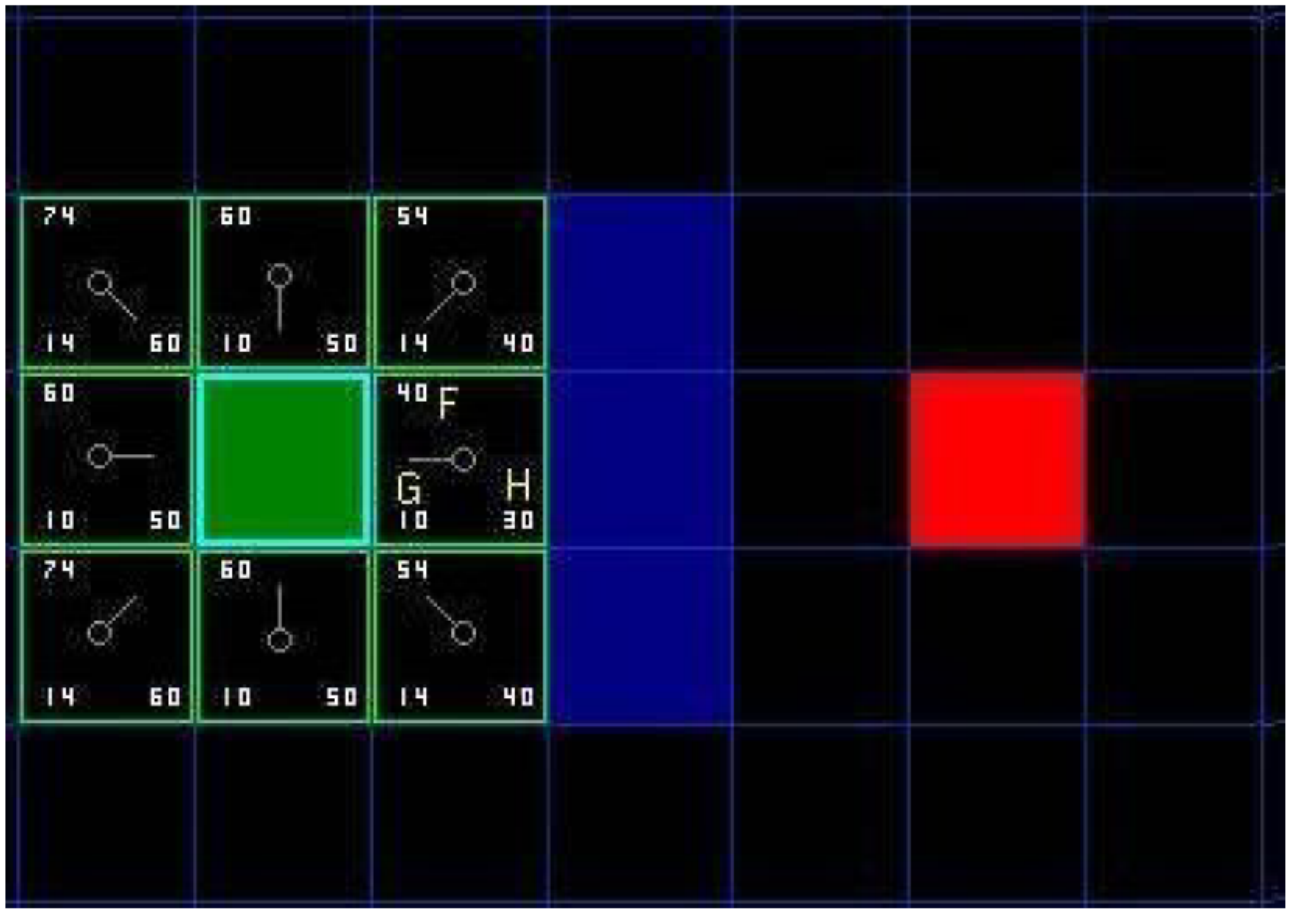
GameManager class

```
// attached to the Camera (a simple ‘follow cam’)

public class GameManager : MonoBehaviour {
   void Start () {
      Camera.main.transform.position = new Vector3(0f, 20f, 20f);
      Camera.main.transform.LookAt(Vector3.zero);
   }
}
```

Pathfinder Class

```
using UnityEngine.AI;

public class Pathfinder : MonoBehaviour {

    private NavMeshAgent nav;

    void Start () {
        nav = GetComponent<NavMeshAgent>();
        nav.destination = transform.position;
    }

    void Update () {
        if (Input.GetMouseButtonDown(0)) {
            Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
            RaycastHit hitInfo;
            if (Physics.Raycast(ray, out hitInfo, 500f)) {
                nav.destination = hitInfo.point;
            }
        }
    }
}
```

A* Pathfinding
(The next few slides are from CT255)



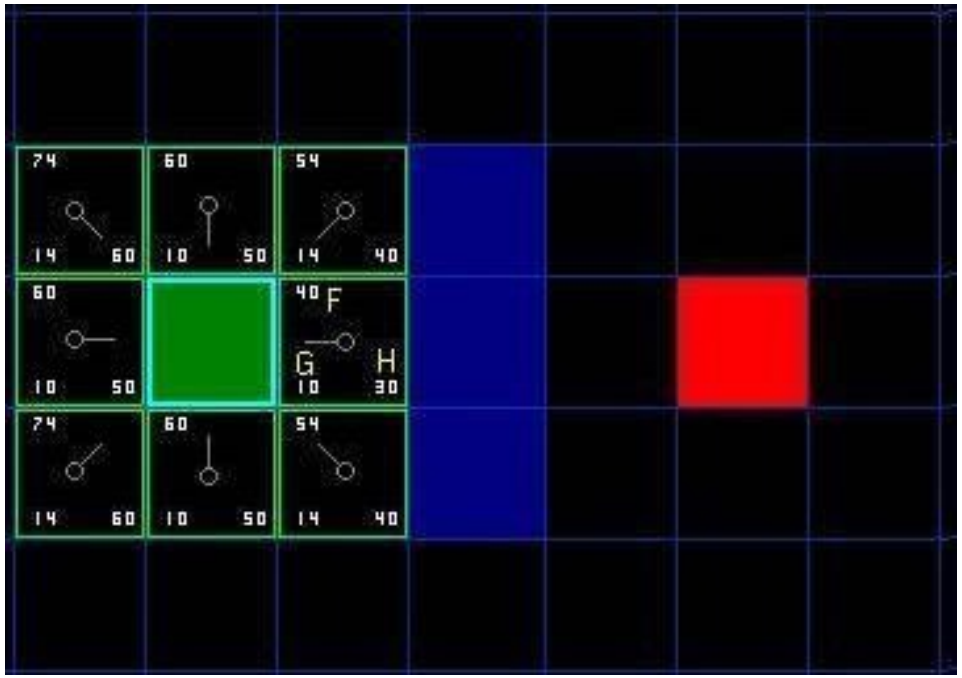
known cost from start to this node

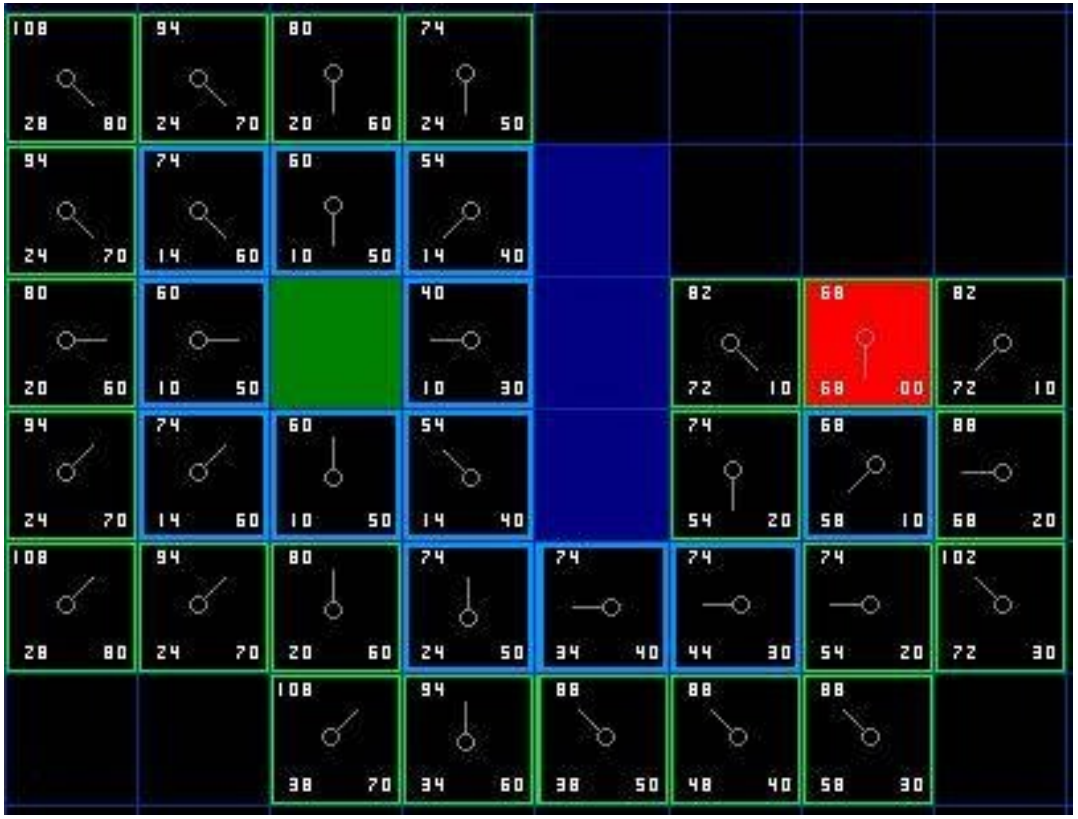
est. cost from this node to goal

$$f = g + h$$

est. cost from start to goal via this node

images from: http://www.policyalmanac.org/games/aStarTutorial.htm


74
14 60
60
10 50
54
14 40
60
10 50
40 F
G 10 H 30
74
14 60
60
10 50
54
14 40


108
28 80
94
24 70
80
20 60
74
24 50
94
24 70
74
14 60
60
10 50
54
14 40
80
20 60
60
10 50
40
10 30
82
72 10
68
68 00
82
72 10
94
24 70
74
14 60
60
10 50
54
14 40
74
54 20
68
58 10
88
68 20
108
28 80
94
24 70
80
20 60
74
24 50
74
34 40
74
44 30
74
54 20
102
72 30
108
38 70
94
34 60
88
38 50
88
48 40
88
58 30

A* Pathfinding

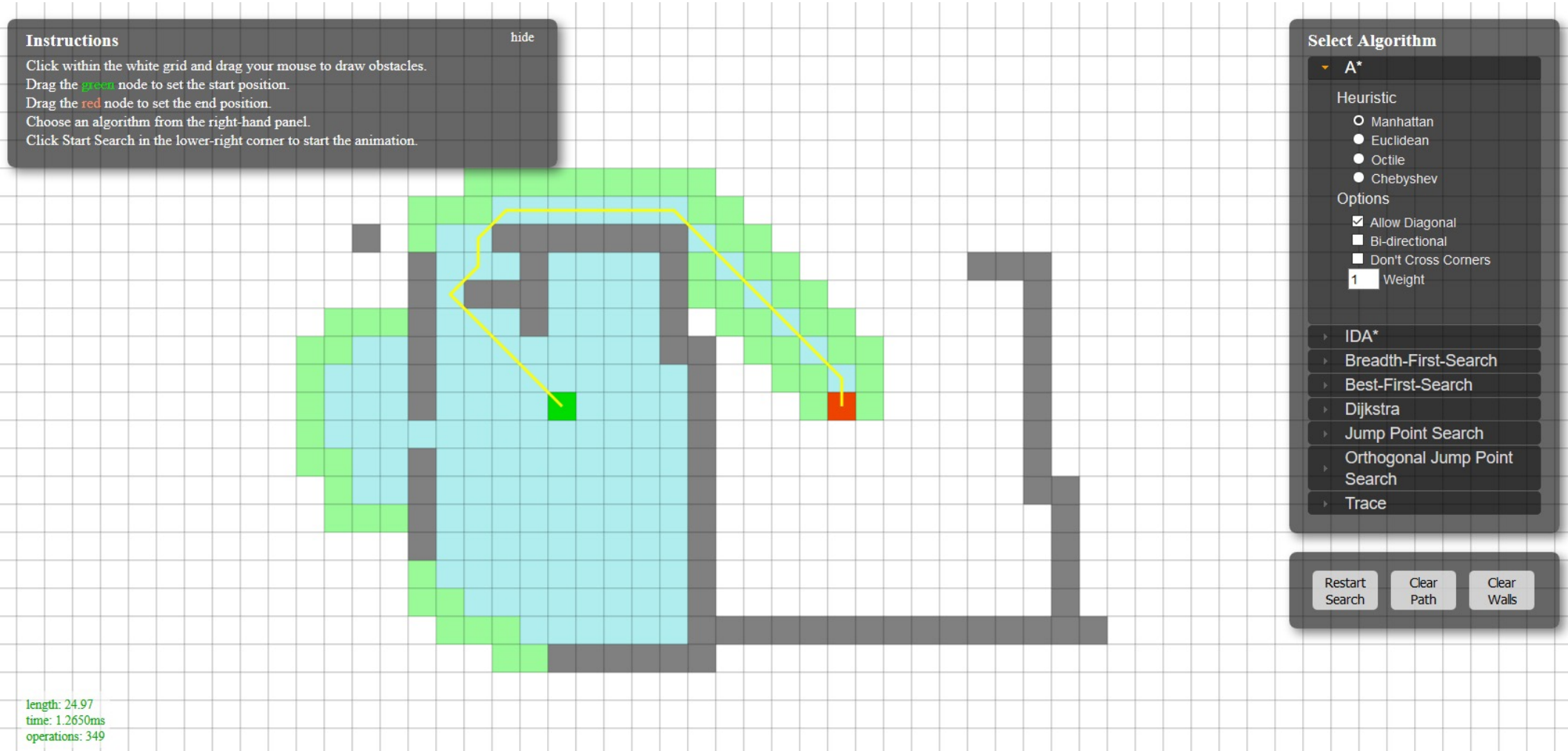
- The fundamental operation of the A* algorithm is to traverse a map by exploring promising positions (nodes) beginning at a starting location, with the goal of finding the best route to a target location.

- Each node has four attributes other than its position on the map:
  - *g* is the cost of getting from the starting node to this node
  - *h* is the estimated (heuristic) cost of getting from this node to the target node. It is a best guess, since the algorithm doesn't (yet) know the actual cost
  - *f* is the sum of *g* and *h*, and is the algorithm's best current estimate as to the total cost of travelling from the starting location to the target location via this node
  - *parent* is the identity of the node which connected to this node along a potential solution path

A* Pathfinding

- The algorithm maintains two lists of nodes, the *open* list and the *closed* list.
- The OPEN LIST consists of nodes to which the algorithm has already found a route (i.e, one of its connected neighbours has been evaluated or *expanded*) but which have not themselves, yet, been expanded.
- The CLOSED LIST consists of nodes that have been expanded and which therefore should not be revisited.
- Progress is made by identifying the most promising node in the open list (i.e., the one with the lowest *f* value) and expanding it by adding each of its connected neighbours to the open list, unless they are already closed.
- As nodes are expanded, they are moved to the closed list.
- As nodes are added to the open list, their *f, g, h* and *parent* values are recorded.
- The *g* value of a node is, of course, equal to the *g* value of its parent plus the cost of moving from the parent to the node itself.

https://qiao.github.io/PathFinding.js/visual/



(PathFinding.js.html)

Implementing A* Pathfinding..

- What data do we need? How might we structure the data?
  - Start loc, target loc
  - Nodes to map the game area (2D array of nodes)
  - Walkable/unwalkable map (2Darray of booleans)
  - Open list (as linked list of nodes?)
  - Storage of final path (as a stack of nodes?)

- What are the initial conditions for this data?
  - Each wall node is unwalkable -> 'closed'
  - All the rest are not open and not closed
  - Calculate f,g,h for starting node and set to 'open'

- What is the general algorithmic step?
  - Find open node with lowest f (call it X)
  - Look at its neighbours: any not closed and not open should become opened: calculate f,g,h and record parent position (i.e. position of X)
  - Close node X

- How will we know when we're finished?
  - If a neighbour is the target, we're done searching
  - If there are no open nodes, the maze is unsolvable

- How will we use what we found in order to have an AI-controlled '*badguy*' chase after a '*player*'?
  - Push target onto stack,
  - Push its parent onto stack
  - Push its parent onto stack
  - Etc.. Until we have pushed start node

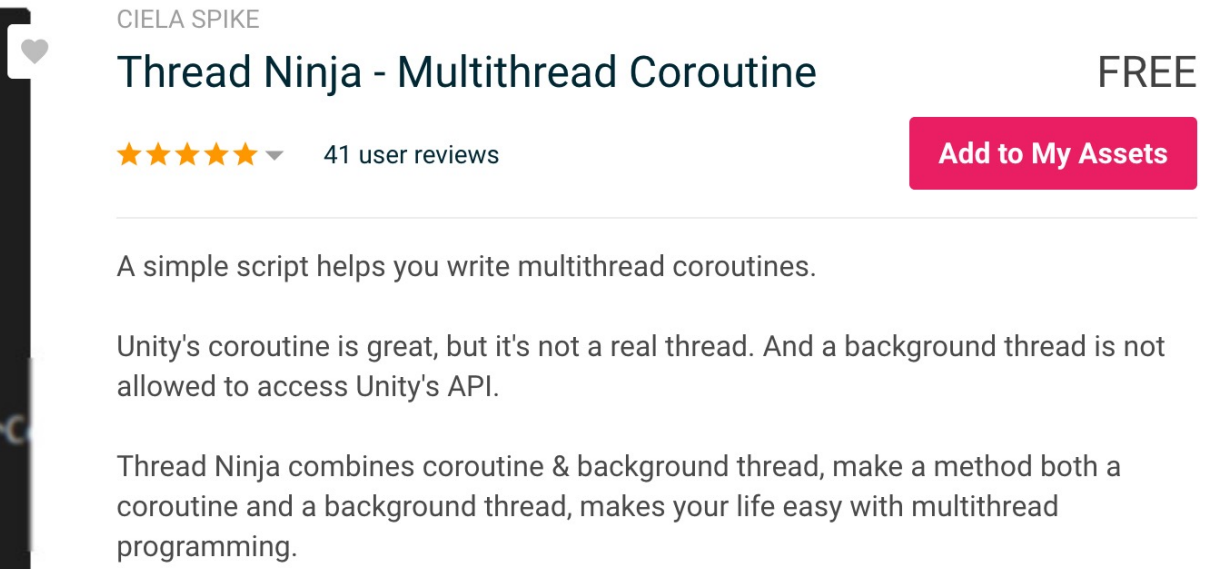
Pathfinding in Demon Pit (1/2)

- Since the DemonPit arena periodically reconfigures (floors drop and rise back, walls rise and drop back), pathfinding can't be performed on a static mesh
- Whenever walls/floor have finished moving, a set of raycasts (at 1x1m intervals) is used to re-determine the walkability of each grid cell. This is carried out by the AStarMesh script, attached to the arena object
- The AI-controlled monsters have the AStarAgent script attached to them, which share use of the single AStarMesh in order to calculate paths

Pathfinding in Demon Pit (2/2)

- A* pathfinding is performed by the AStarAgent in a thread, using the free asset “Thread Ninja” from the asset store, which simplifies C# threads
- The AStarMesh is locked while an agent is using it, so other agents will potentially be delayed waiting for it, for a few frames
- In another game I’m working on (with much larger maps than Demon Pit), I have implemented a pool of AStarMeshes, each having their own set of Nodes. This allows multiple agents to simultaneously calculate paths

```
IEnumerator AsyncCoroutine()
{
    // On background thread.
    DoHeavyComputing();

    // Jump to Unity main thread.
    yield return Ninja.JumpToUnity;
    CallSomeUnityAPI();
    yield return StartCoroutine(AnotherC

    // Jump to background.
    yield return Ninja.JumpBack;
    DoHeavyComputing();
}
```



Pathfinding in Demon Pit

- See separate document for code:
  - AStarMesh.cs
  - AStarAgent.cs
  - Relevant code from Monster.cs

  - NB this is relatively advanced so don't be concerned if you can't follow it. It's definitely not examinable material for this module, but hopefully it's a useful example nevertheless.