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 gridbased approach we take below => more efficient to use

The licenchy Shade Came Shade 20 20 20 20 20 20 20 20 20 20 20 20 <th>🖑 🕈 S 🗷 🛄</th> <th>Center</th> <th>🗊 Local</th> <th></th> <th>l</th> <th></th> <th></th> <th>🕑 Colla</th> <th>b 🔹 🙆 Accou</th> <th>nt 🔹 Layers 🗣</th> <th>Layout 🔹</th>	🖑 🕈 S 🗷 🛄	Center	🗊 Local		l			🕑 Colla	b 🔹 🙆 Accou	nt 🔹 Layers 🗣	Layout 🔹
Create Stade 120 1 Carae Cara	≔ Hierarchy	🔒 🚛 🛛 🗰 Scer	e Game	🛱 Asset Store	🎥 Animator		*≡	O Inspector	Navigation		*=
Image: Canadity in the component workflow. Wain Canadity in the component workflow. <tr< td=""><td>Create * Q*All</td><td>Shaded</td><td>* 2D</td><td>⊲)) 🖬 💌</td><td></td><td>Gizmos * Q*All</td><td></td><td>Agents</td><td>Areas</td><td>Bake</td><td>Object</td></tr<>	Create * Q*All	Shaded	* 2D	⊲)) 🖬 💌		Gizmos * Q*All		Agents	Areas	Bake	Object
Mar Camera Directional Light Vilor Wall Wall Wall Wall Wall Wall CleverCube Propect Console	▼ 🚭 Main*	*=					y 🔒		, incus	Durce	005/000
Project	Main Camera Disectional Links							Learn instead about	the component workfl	low.	
Wal Wal Wal Wal Wal Wal CleverCube CleverCube Definition of the state	Eleor							Baked Agent Size			
Wai Wai Wai Wai Wai CleverCube CleverCube	Wall									R = 0.5	
Wall Wall Wall Wall Wall CleverCube Image: CleverCube <	Wall										
Wall Wall Wall CleverCube Image: CleverCube	Wall						< Persp			/	1
Wall Wall CleverCube 0.4 Output 45° Agent Radius 0.5 Agent Height 2 Max Slope 45° Gerrated Off Mesh Links 0.4 Drop Height 0 Jump Distance 0 Output Advanced Project Console	Wall									H = 2	
Vall CleverCube CleverCube Vall CleverCube Vall CleverCube Vall CleverCube Vall CleverCube Vall CleverCube Vall Va	Wall							0.4			
LeverCube Agent Radius Agen	Wall									45°	
Agent Radius 0.5 Agent Radius	CleverCube										
Agent Height 2 Max Slope Step Height 0.4 Cenerated Off Mesh Links Torop Height 0 Ump Distance 0 Advanced Project Console								Agent Radius	0.5		
Advanced Project ■ Console								Agent Height	2		
Sep Height 0.4 Generated Off Mesh Links Drop Height 0 Jump Distance 0 Advanced Project © Console								Max Slope		0	45
Project Console								Step Height	0.4		
Navmesh Display Show NavMesh Show HeightMesh Project								Generated Off Mes	h Links		
Navmesh Display Show NavMesh Show HeightMesh Project								Drop Height	0		
Navmesh Display Show NavMesh Show HeightMesh ■ Project □ Console								Jump Distance	0		
Navmesh Display Clear Bake Show NavMesh Show HeightMesh Image: Clear								Advanced			
Navmesh Display Show NavMesh Show HeightMesh								, , , , , , , , , , , , , , , , , , ,		Clear	Bake
Show NavMesh Show HeightMesh						Navmesh D	Display			Clear	
Show HeightMesh						Show NavMesh					
Project 🗄 Console						Show HeightMesh					
	B Project										
					Q						

The CleverCube Object

	🖉 Collab 🔹 🔼	Account • Layers •	Layout 🔹
ore 💏 Animator 📲	🟮 Inspector 🛛 🔀 Navigation	1	<u></u> →=
Gizmos * Q*All	CleverCube		Static 🔻
У		+ Laver Default	
			<u> </u>
	Mesh	Cube	
			~
< Persp	Box Collider		<u> </u>
	Mesh Renderer		<u> </u>
	A gent Type	Humanoid	¥•
	Rase Offset	0.5	•
	Base Offset	0.5	
	Steering		
	Speed	3.5	
	Angular Speed	120	
	Acceleration	8	
	Stopping Distance	0	
	Auto Braking		
	Obstacle Avoidance		
	Radius	0.5	
	Height	2	
	Quality	High Quality	†
	Priority	50	
	Path Finding		
a •=	Auto Traverse Off Mesh Link	\checkmark	
Q 4 V *	Auto Repath		
	Area Mask	Everything	\$
	🔻 🕡 🗹 Pathfinder (Script)		a a ,
	Script	@ Pathfinder	0

- 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)



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

74 60 54 14 60 10 10 50 14 60 40 60 40 60 40 60 40 60 40 60 40 60 40 60 40 60 60 60 54 60 54 60 54 60 54 60 10 50 14					
14 60 10 50 14 40 60 40 F F F F 0 60 60 F F 10 50 54 F F 74 60 54 F F 14 60 10 50 14 40	74 Q	60 Q	54		
	14 60 60	10 50	14 40 40 F		
74 50 54 54 14 50 10 50 14 40			Ģ H		
14° 60 10 50 14 040	74	50 	54		
	14 60	10 50	CH 240		

108	94	80	74				
28 80	24 78	20 60	24 50				
94	74	60	54				
24 70	14 60	10 50	14 40				
80	60		40		82	68	82
<u> </u>	0—		0		٩	Ŷ.	٩
20 60	10 50		10 30		72 10	68 00	72 10
	74	•	54		24 Υ	58 	0
24	140 60	10 50	<u> 14 (40</u>		54 20	58 10	68 20
6	94 °	".	74 	74. 	74 0	74 —O	102
28 80	24 70	20 60	24 50	34 40	44 30	54 20	72 30
		·•• ८	94 	**	**	**	
		38 70	34 60	38 50	48 40	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

•

•

•

٠

٠

- 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



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.