Nav Mesh

Objective
To show how to use nav mesh to allow players and enemies to move around a complex map

Note
If you give the students the RTS camera script to use with this project they can do a lot more with it.

First Scene

Basic setup of a NavMesh.

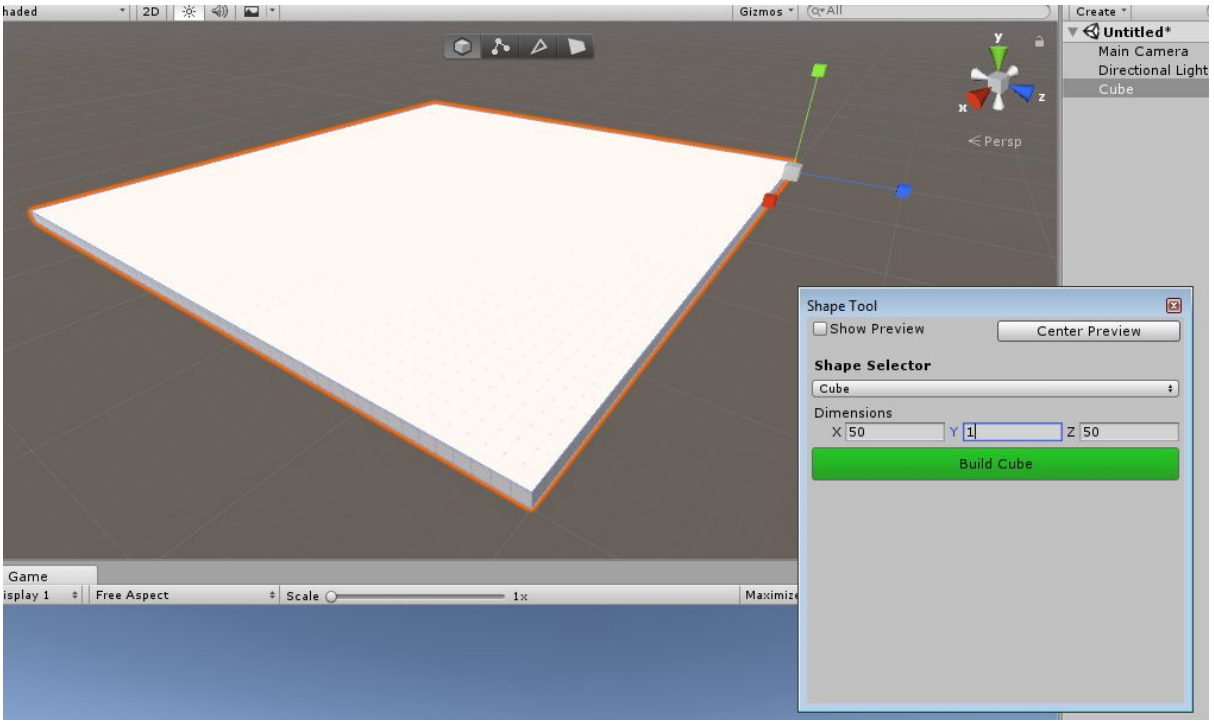
Setup
- Create a new scene and save it
-

Lesson

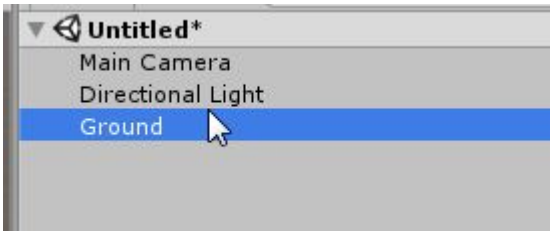
Start a new Project

First thing to do is to save a new scene.

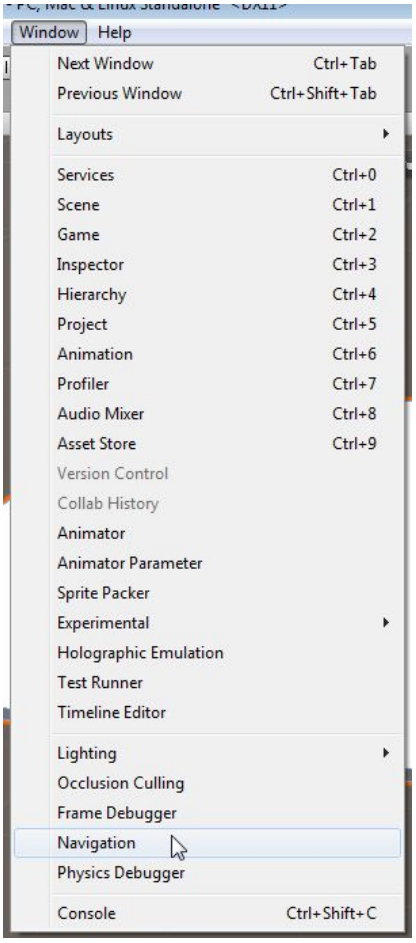
Using Probuilder create a flat cube**, 50 X 1 X 50.**



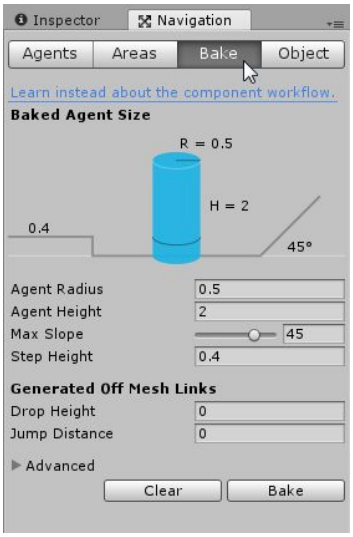
Rename this shape **‘Ground’**



Click on **Window -> Navigation**

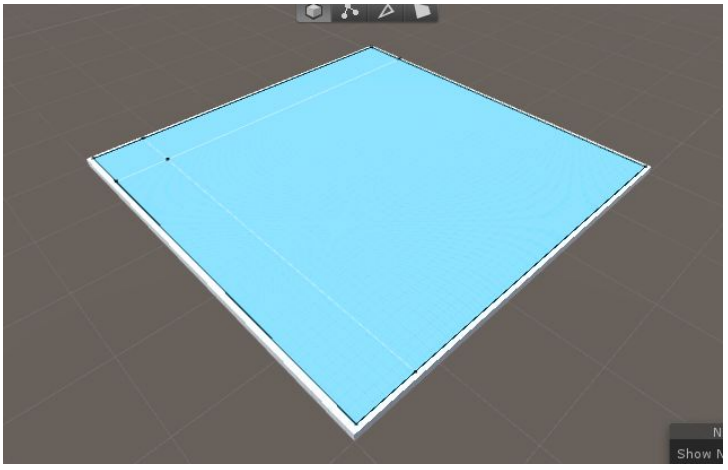


In the navigation window select the Bake tab



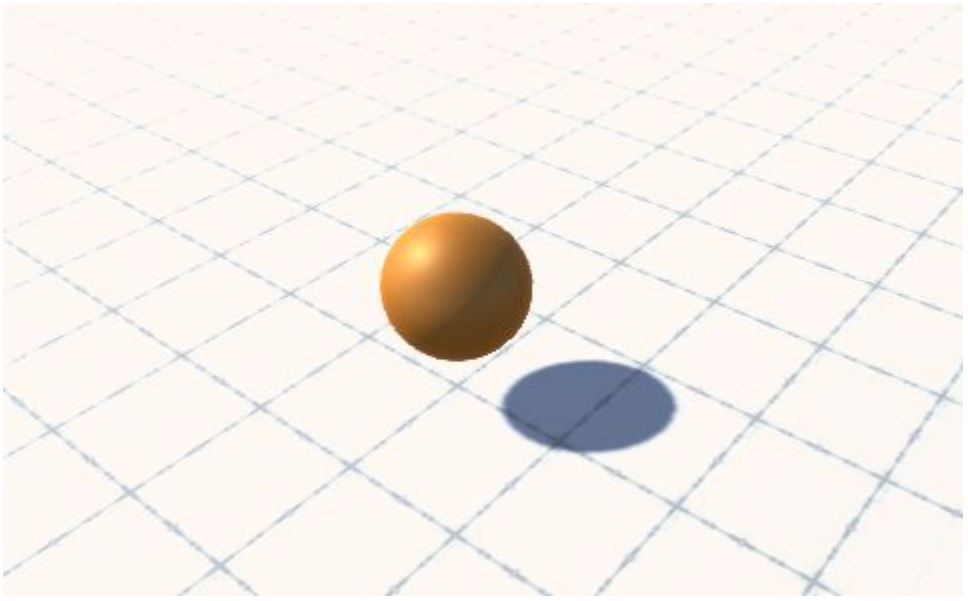
Press the **‘Bake’** button

When you do it will create what is called a **Nav Mesh** on your ground object.

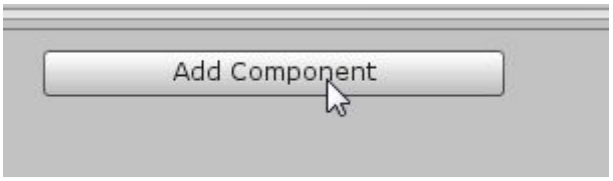


Now let’s create enemies to travel on our navmesh.

Create a Unity Sphere game object and place it in the middle of your scene. Name it ‘Enemy’ and give it a new material to make it stand out.



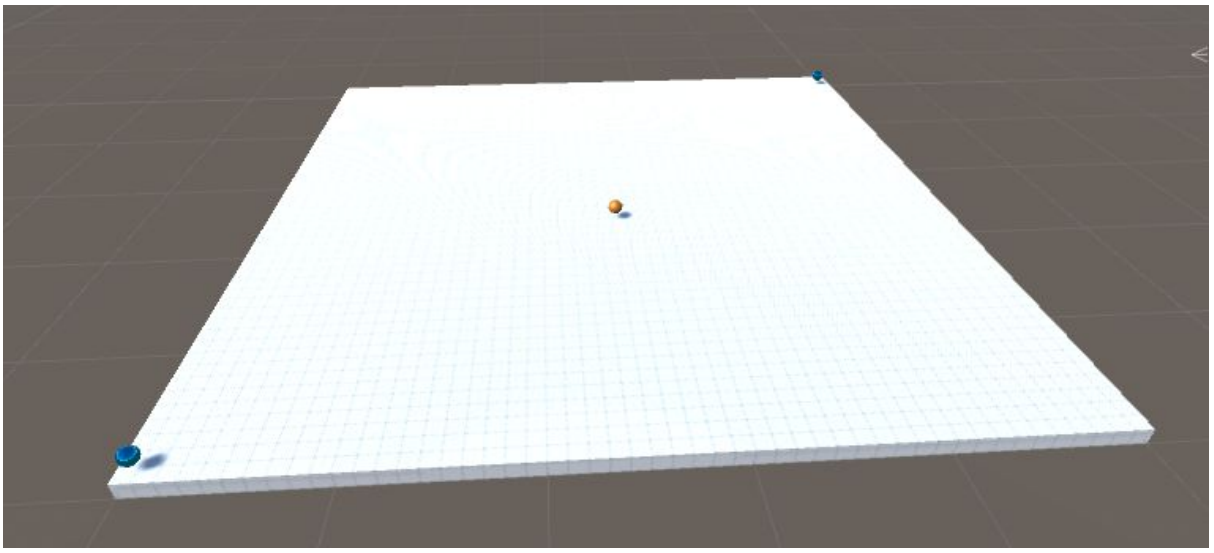
With the Enemy object and in the inspector click the '**Add Component'** button



Add a '**Nav Mesh Agent**' component to it

This makes the enemy now work with the nav mesh we have created.

Create two more spheres, remove their colliders and name them, Start node, End Node and place them at different corners of the ground object you have created.



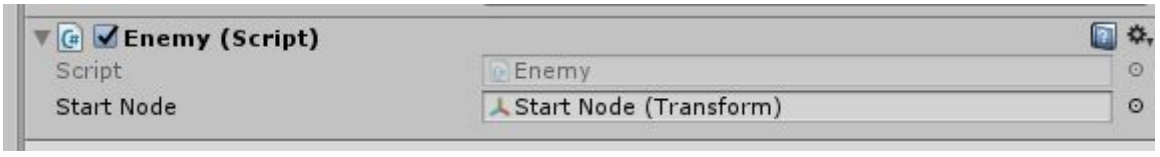
Create a **Scripts** folder

And create a new script called **‘Enemy’** and place it on the enemy GameObject

Work through the group

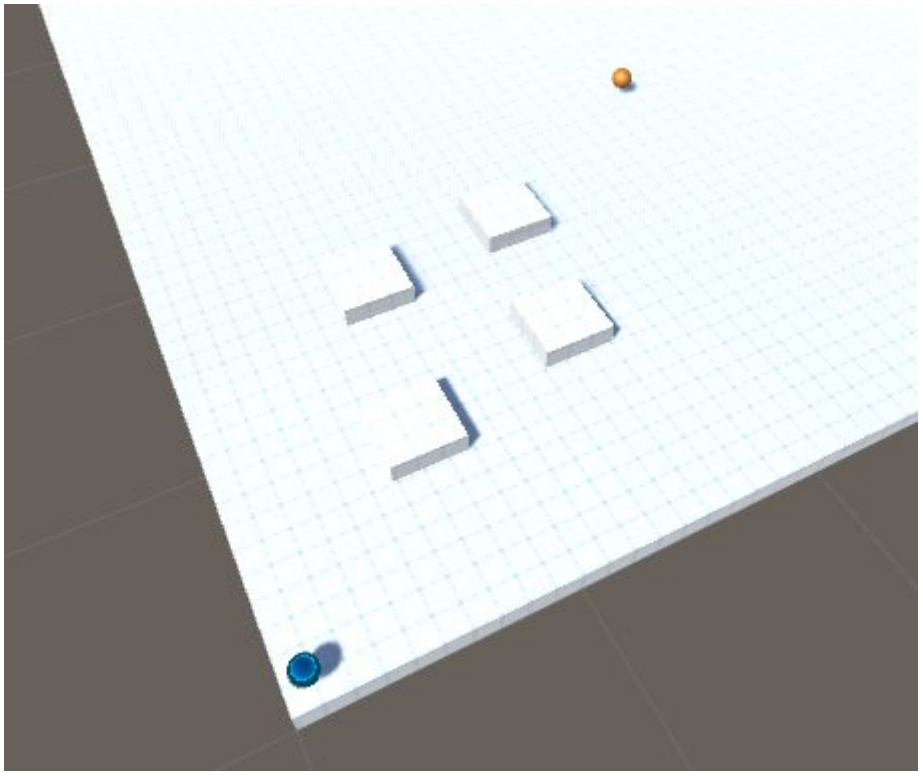
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.AI;

public class Enemy : MonoBehaviour {

    public Transform startNode;
    private NavMeshAgent NMA;

    // Use this for initialization
    void Start()
    {
        NMA = GetComponent<NavMeshAgent>();
        NMA.destination = startNode.position;
    }

}
```

Drag the Start Node object to this script

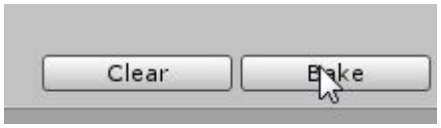


Press Play and see what happens

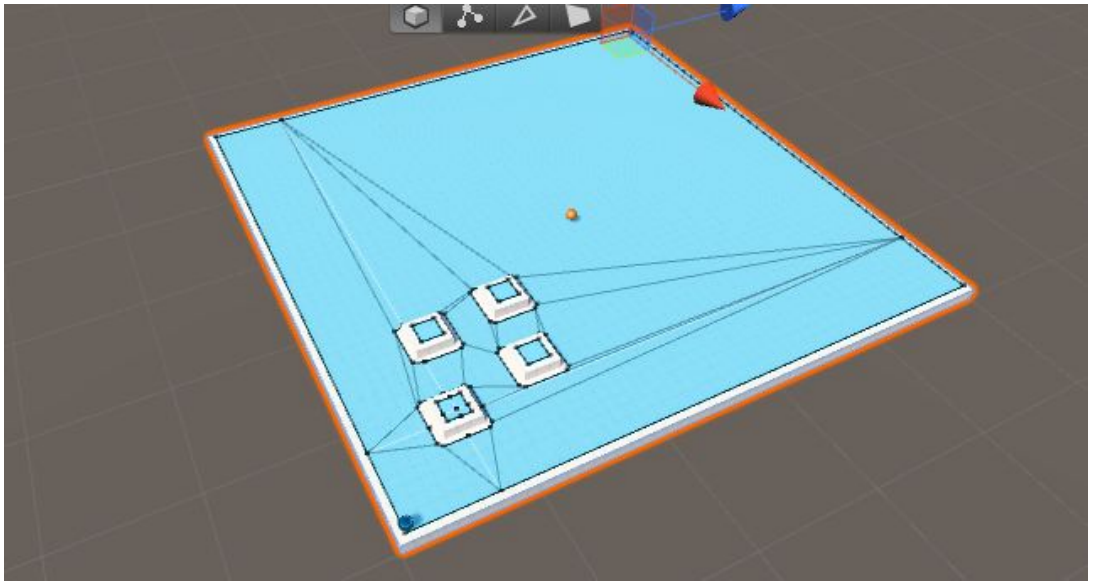
Add some new Probuilder squares to the ground between the enemy and its node.



Select the **navigation** panel and select the ground object and press **‘Bake’** again



Notice that the navmesh now cuts out the new obstacles
(*If not, select the cubes and in the Navigation panel → Object tab tick Navigation Static)*



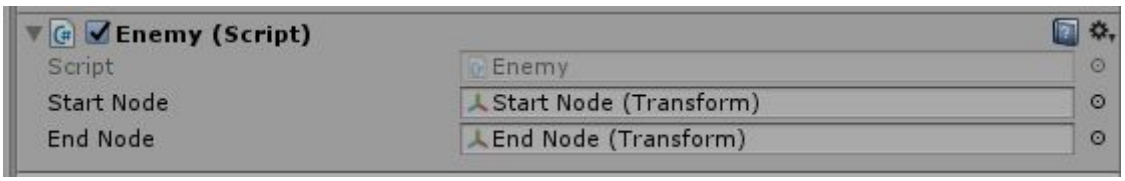
Press **Play** again.

Notice that our enemy finds the shortest path around out new scene

Update the Enemy Script

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.AI;

public class Enemy : MonoBehaviour {

    public Transform startNode;
    public Transform endNode;

    private NavMeshAgent NMA;

    // Use this for initialization
    void Start()
    {
        NMA = GetComponent<NavMeshAgent>();
        NMA.destination = startNode.position;
    }

    // Update is called once per frame
    void Update()
    {

        if (Vector3.Distance(transform.position, startNode.position) < 2)
        {
            NMA.destination = endNode.position;
        }

        if (Vector3.Distance(transform.position, endNode.position) < 2)
        {
            NMA.destination = startNode.position;
        }

    }


}
```

Then drag the End Node into the inspector



Press Play again and see what happens.

Duplicate the Enemy Object and then watch them walk back and forth. (the script won't allow a prefab yet)

Results

The units can now walk back and forth around objects easily.

Make a Diablo style game

Make a game where we can control the player using raycasts and Nav Meshes

Setup

- Create a new scene
- Make a capsule and put a nav mesh agent on it
- Name the capsule 'Player'
- Tag the player with the 'Player' tag
- Make some ground objects to walk around
- Make the ground mesh static
- Make a 'Ground' Tag and tag the ground with it
- Bake a nav mesh onto the ground

- Make a script called 'GameController' and place it on the camera
- Make a script called 'PlayerController' and place it on the capsule

- Make a Sphere, scale it to 0.5 in all directions, remove it's collider and name it 'Movement Marker'

The provided RTS camera script is good for this game.

Lesson

First work on the GameController script

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class GameController : MonoBehaviour
{

    public GameObject movementMarker;

    void Update()
    {

        RaycastHit hit;
        Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
        float distanceOfRay = 100;
        if (Physics.Raycast(ray, out hit, distanceOfRay))
        {
            movementMarker.transform.position = hit.point;

        }
    }
}
```

You will need to move the Movement marker sphere onto the public GameObject.

When run the marker should follow the mouse pointer around the screen when it is over an object.

Now let's make it only work when it's over a object marked as 'Ground' and only when the player clicks the left mouse button

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class GameController : MonoBehaviour
{

    public GameObject movementMarker;

    void Update()
    {

        RaycastHit hit;
        Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
        float distanceOfRay = 100;
        if (Physics.Raycast(ray, out hit, distanceOfRay))
        {
            if (Input.GetMouseButton(0))
            {
                if (hit.transform.tag == "Ground")
                {

                    movementMarker.transform.position = hit.point;
                }
            }
        }
    }
}
```

Now that we have a position that is valid, let's send this info to the player navMeshAgent and get them to walk there.

Now Let's open the player Script

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.AI;

public class PlayerController : MonoBehaviour
{

    NavMeshAgent agent;

    void Start()
    {
        agent = GetComponent<NavMeshAgent>();
    }

    public void MovePlayer(Vector3 newPosition)
    {
        agent.SetDestination(newPosition);
    }
}
```

Note that we have made the function ‘MovePlayer’ public so we can access it from the other script.

Back to the GameController script

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class GameController : MonoBehaviour
{

    public GameObject movementMarker;
    public GameObject player;

    void Update()
    {

        RaycastHit hit;
        Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
        float distanceOfRay = 100;
        if (Physics.Raycast(ray, out hit, distanceOfRay))
        {
            if (Input.GetMouseButton(0))
            {
                if (hit.transform.tag == "Ground")
                {

                    movementMarker.transform.position = hit.point;

                    player.GetComponent<PlayerController>().MovePlayer(hit.point);

                }
            }
        }
    }
}
```

Make sure you drag the player Capsule to the player GameObject in this script.

Now when you press play you should be able to click somewhere and the player will move there.

Results

Students should be now able to control a player using a nav mesh and raycasts via the mouse

Making Doors

Let's use triggers to make doors that open when the player enters them.

Setup

Same as previous lesson

- Add another Cube and set it to be a trigger
- Use the Trigger display script from the previous triggers lesson
- Make a Scripts Called ‘DoorTrigger’ and place it on the trigger

- Put a cube called ‘Door’ blocking the path somewhere
- If you are using Probuilder, set the door to ‘Mover’
- Put a NavMeshObstacle Component on it.
- Change the setting on the NavMeshObstacle to ‘Carve’

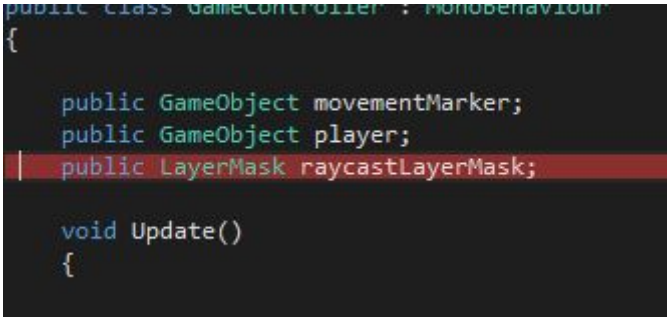
- To make our player work with triggers we need to give it a rigid body.
- Due to weirdness, tick all the constraints in the rigid body (all rotation and positions)

Lesson

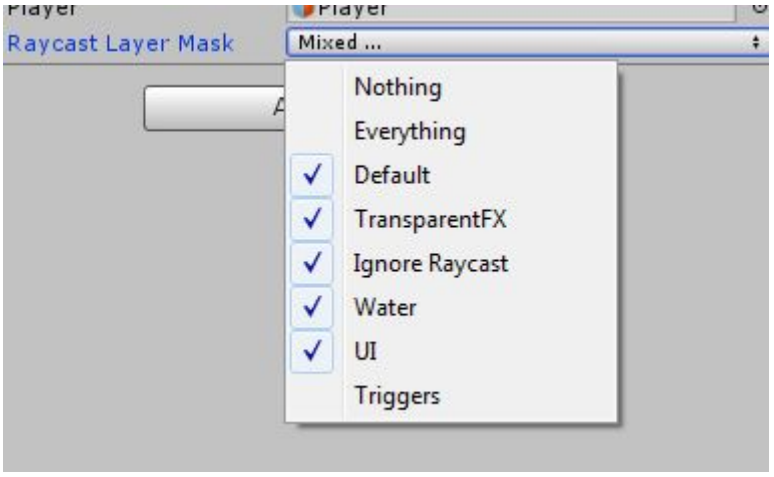
Demonstrate to the class how NavMeshObstacle’s work. Move the door around and show the player can't get past it.

Also not that the Trigger box blocks out raycast. So we need to make a mask for it.

Add this to the top of the GameController script;



Notice that now in the inspector we can select layers on and off.
Select all the layers except our new Triggers layer.



Now we just need to update our raycast.

Change this line

```
void Update()
{

    RaycastHit hit;
    Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
    float distanceOfRay = 100;
    if (Physics.Raycast(ray, out hit, distanceOfRay, raycastLayerMask))
    {
        if (Input.GetMouseButton(0))
        {
            if (hit.transform.tag == "Ground")
            {
```

Now show that the raycast is now not blocked by the trigger box.

Open the ‘DoorTrigger’ script

Add this

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class DoorTrigger : MonoBehaviour
{

    public GameObject doorToOpen;

    void OnTriggerEnter(Collider other)
    {
        if (other.tag == "Player")
        {
            Debug.Log("Open!");

            doorToOpen.SetActive(false);
        }
    }

    void OnTriggerExit(Collider other)
    {
        if (other.tag == "Player")
        {
            Debug.Log("Close!");

            doorToOpen.SetActive(true);
        }
    }

    void OnDrawGizmos()
    {

        if (doorToOpen)
        {
            Gizmos.color = Color.blue;
            Gizmos.DrawLine(transform.position, doorToOpen.transform.position);
        }
    }

}
```

You will need to drag the door that you want it to trigger to the script.

Results

The players should now be able to open doors with their players

Adding in the Enemies

Now that we have a player walking around we need to add enemies.

Setup

Same as previous lesson

- Add another capsule and colour it red
- Name this capsule ‘Enemy’
- Add a NavMeshAgent to the Capsule
- Make a script called ‘DiabloEnemy’ and place it on the Capsule

Lesson

With the ‘Enemy’ Script

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.AI;

public class DiabloEnemy : MonoBehaviour
{

    public float detectionRange = 12;
    public GameObject player;
    private NavMeshAgent navAgent;

    // Use this for initialization
    void Start()
    {
        player = GameObject.FindGameObjectWithTag("Player");
        navAgent = GetComponent<NavMeshAgent>();
    }

    // Update is called once per frame
    void Update()
    {

        if (Vector3.Distance(transform.position, player.transform.position) < detectionRange)
        {
            navAgent.destination = player.transform.position;
        }

    }
}
```

Now the enemy will sit there doing nothing until the player gets within 12 units of them
At that point they will chase the player.

Get them to change the speed values and colours of the enemies and prefabs them.so they can be made easier.

Results

You should now be able to make a map where they open doors and have different enemies they can place.

Attacking

Using timers and triggers let's program the player attacking.

Setup

Same as previous lesson

- Make a sphere
- Name the Sphere Bullet
- Change it's scale to 0.5,0.5,0.5
- Make its collider a trigger
- Give it a funky material
- Give the sphere a trail render
- Give the train render a material (Particle shader)
- Change the trail render's Time to 0.5
- Change the trail render's Width to 0.5
- Give the sphere a RigidBody
- Turn off Gravity on the RigidBody
- Make a script called 'Bullet' and place it on the sphere
- Make a prefab of the bullet
- Delete the bullet in the scene

- Add a Rigidbody to the enemy prefabs
- Constrain all movement and rotation on these rigidbodies

Lesson

Update the Player Controller script

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.AI;

public class PlayerController : MonoBehaviour
{

    public float timeBetweenAttacks = 0.2f;
    public GameObject bulletPrefab;
    public float bulletSpeed = 5f;
    public bool canShoot = true;

    NavMeshAgent agent;

    void Start()
    {
        agent = GetComponent<NavMeshAgent>();
    }

    public void MovePlayer(Vector3 newPosition)
    {
        agent.SetDestination(newPosition);
    }

    public void Attack(Vector3 attackLocation)
    {
        if (canShoot)
        {
            GameObject GO = Instantiate(bulletPrefab, transform.position,
                Quaternion.identity) as GameObject;
            GO.GetComponent<Rigidbody>().AddForce((attackLocation - transform.position)
                * bulletSpeed, ForceMode.Impulse);

            Destroy(GO, 3f);
            canShoot = false;
            Invoke("ResetShooting", timeBetweenAttacks);
        }
    }

}
```

Then add one more function to it

```
private void ResetShooting()
{
    canShoot = true;
}
```

make sure you drag the bullet prefab to the bullet prefab slot

Now let’s update the GameController script to call this public function

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class GameController : MonoBehaviour
{

    public GameObject movementMarker;
    public GameObject player;
    public LayerMask raycastLayerMask;

    void Update()
    {

        RaycastHit hit;
        Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
        float distanceOfRay = 100;
        if (Physics.Raycast(ray, out hit, distanceOfRay, raycastLayerMask))
        {
            if (Input.GetMouseButton(0))
            {
                if (hit.transform.tag == "Ground")
                {

                    movementMarker.transform.position = hit.point;

                    player.GetComponent<PlayerController>().MovePlayer(hit.point);

                }
            }
            else if (Input.GetMouseButton(1))
            {
                if (hit.transform.tag == "Enemy")
                {
                    Debug.DrawLine(player.transform.position,
                        hit.transform.position, Color.yellow, 0.1f);
                    player.GetComponent<PlayerController>().Attack(hit.transform.position);
                }
            }
        }
    }
}
```

You should now be able to shoot bullets at the enemies when you right click on them;

Let's make the bullets do damage;

Open up the ‘Bullet’ Script

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Bullet : MonoBehaviour {

    public float damage = 2;

    void OnTriggerEnter(Collider other)
    {
        Debug.Log(other.tag);
    }

}
```

Now when you shoot an enemy it will show its tag in the Debug window

Now open up the DiabloEnemy Script

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.AI;

public class DiabloEnemy : MonoBehaviour
{

    public float health = 6;

    public float detectionRange = 12;
    public GameObject player;
    private NavMeshAgent navAgent;

    // Use this for initialization
    void Start()
    {
        player = GameObject.FindGameObjectWithTag("Player");
        navAgent = GetComponent<NavMeshAgent>();
    }

    // Update is called once per frame
    void Update()
    {

        if (Vector3.Distance(transform.position, player.transform.position) < detectionRange)
        {
            navAgent.destination = player.transform.position;
        }

    }

    public void TakeDamage(float damage)
    {
        health = health - damage;

        if (health <= 0)
        {
            Destroy(this.gameObject);
        }
    }

}
```

Now we have a way to pass damage to the enemy let’s update our bullet script

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Bullet : MonoBehaviour {

    public float damage = 2;

    void OnTriggerEnter(Collider other)
    {
        if (other.tag == "Enemy")
        {
            other.GetComponent<DiabloEnemy>().TakeDamage(damage);
            Destroy(this.gameObject);
        }

    }

}
```

Now the enemies should die when you shoot them

Results

You should now be able to shoot and kill any enemies they have in their game.

Further tasks for students to work through

- Build a more complex level to walk around
- Make some enemies fire back at the player and kill the player.
- Make the doors swivel open instead of deactivating
- Add sounds and particles to the game.