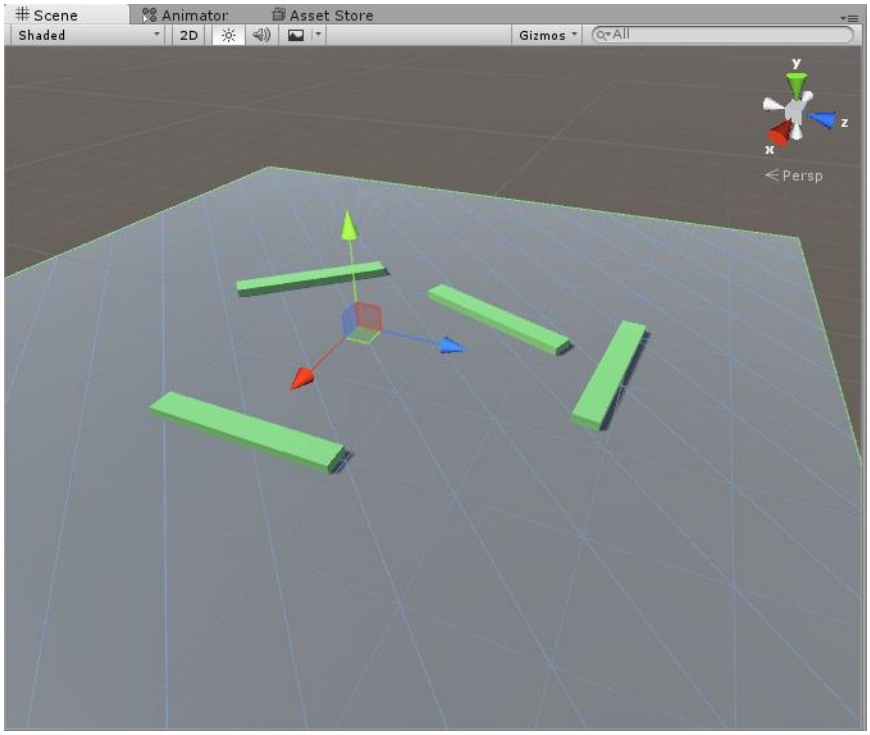
Tutorial – Character Controllers

In this tutorial we'll step through how to create the scene in the Character Controller Sample available on perforce.

By the end of this tutorial you should have a scene containing a player controllable character that uses the Character Controller component, and an 'enemy' that follows the player using a navigation mesh.

Setting up the Scene:

Set up a simple scene so that you have a floor (use a 3D plane) and some obstacles (some stretched cubes are fine).



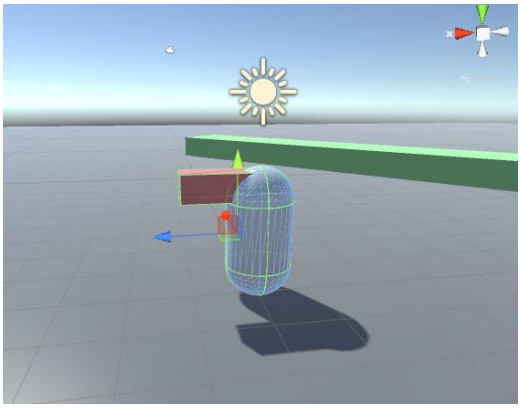
Put box collider components on everything.

That's it. This will be our test scene. It will be enough to test our character controller with.

Create the Player:

Create a 3D Capsule and position it in the scene. Make sure that the capsule isn't intersecting the floor.

This capsule will be our 'player'. To ensure you know which direction your player is facing, you might want to add a cube or other shape as a child of the capsule.



Add a Capsule Collider component and the Character Controller component to the capsule.

You can use the default parameters for these components. After this tutorial you could try modifying the values in the Character Controller component to see what effect they have on the player's movement.

This completes the set for the player.

The last thing to do is to add the script that will pass the keyboard input to the Character Controller in order to move the player. This script was shown in the lecture slides for this session, and is shown again below.

Create a new script in the Unity editor called *CharacterMovement* and enter the following code.

```
using UnityEngine;
using System.Collections;

public class CharacterMovement : MonoBehaviour {

    public float MoveSpeed = 0.5f;
    public float RotateSpeed = 1f;
    CharacterController cc;

       // Use this for initialization
       void Start () {
        cc = GetComponent<CharacterController>();
       }

       // Update is called once per frame
       void Update () {
        // rotate the character according to left/right key presses
        transform.Rotate(Vector3.up * Input.GetAxis("Horizontal") * RotateSpeed);
        // move forward/backward according to up/down key presses
        cc.Move(transform.forward * Input.GetAxis("Vertical") * MoveSpeed);
        // apply gravity
        cc.SimpleMove(Physics.gravity);
       }
}
```

Let's break this down.

In the *Start* function all we are doing is getting a reference to the Character Component attached to the game object and storing this in the variable *cc*. We'll need this variable during the *Update* function.

In the *Update* function we rotate the game object's transform according to left/right keypresses. The piece of code that gets the key presses is `Input.GetAxis("Horizontal")`. We then multiply this by the rotation speed, and this value is multiplied by the 'up' vector – the result being that the game object is rotated around the Y-axis according to how much the left and right arrow keys have been pressed.

The game object is moved using a similar approach. The amount of movement is calculated using the 'vertical' keys (up and down arrows) and multiplied by the transform's forward vector. This has the effect of moving the object forward no matter which direction it is facing. Finally, we pass the result to the *Move* function of the *CharacterController* to move our player. The *CharacterController* will check for any collisions and respond accordingly.

Finally we apply gravity to our player by calling the *SimpleMove* function and passing in the *Physics* gravity variable.

Save your script and return to the editor. If you play your game now you should be able to move your player around the scene using the arrow keys. Adjust the rotation and move speeds until your character feels right.

Create the Navigation Mesh:

Before we can create the enemy we need to create the navigation mesh that the enemy will use to figure out how to get to the player.

Display the *Navigation* window by selecting *Window* -> *Navigation*.

In the *Hierarchy* window select the floor. Some properties for the floor will display in the *Navigation* window. Set the floor to *Static* and *Walkable*.

The navigation mesh only works on game objects that won't move. But Unity can't tell which objects will move and which won't, so we need to let it know that the floor won't move by checking the *Static* option. We also need to tell Unity that characters can walk on the floor by setting the *Navigation Area* variable to *Walkable*. (Anything you don't want your enemies to walk on, like walls, will be set to *Not Walkable*).

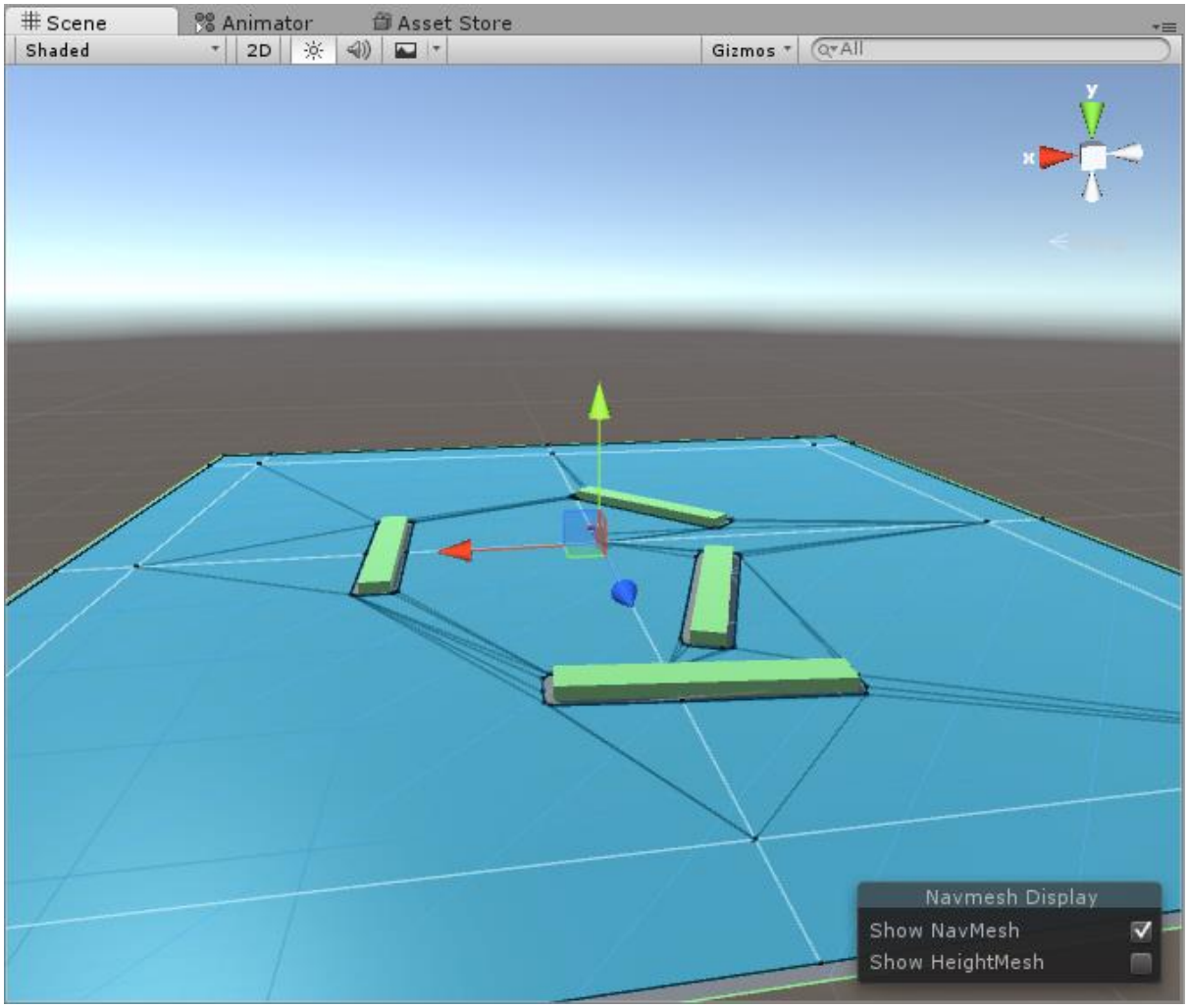
Now select all of the obstacles and set these to *Static* and *Not Walkable*.

Once this is complete, look at the bottom of the *Navigation* window. You should see two buttons there: *Clear* and *Bake*.

Press the *Bake* button.

What this does is it takes the navigation settings (walkable and non-walkable areas) for all the objects in your game and generates a 'navigation mesh'. Your enemy will be able to walk anywhere on this mesh, and when we give the enemy a location to move to it will use this mesh to calculate the best way to get there.

After the *Bake* button has finished processing you should see the navigation mesh appear in blue in your scene. (This will disappear as soon as you close the Navigation window).



Create the Enemy:

Now we come to the final part, creating an enemy that will follow the player.

Add another 3D Capsule to your scene. Give it a Capsule Collider and this time instead of adding a *Character Controller*, add a *Nav Mesh Agent* component.

The *Nav Mesh Agent* uses the *Nav Mesh* to move the enemy around the scene. The default values are fine for now.

The final step is to add a script that will tell the enemy where to go (i.e., to follow the player).

Create a new script called *FollowPlayer*, attach it to your enemy, and then edit the code as follows:

```
using UnityEngine;
using System.Collections;

[RequireComponent(typeof(NavMeshAgent))]
public class FollowPlayer : MonoBehaviour {

    public Transform follow;

    NavMeshAgent agent;
    float timer = 0;

       // Use this for initialization
       void Start () {
           agent = GetComponent<NavMeshAgent>();
       }

       // Update is called once per frame
       void Update () {
        if (timer <= 0)
        {
            agent.SetDestination(follow.position);
            timer = 1;
        }
        timer -= Time.deltaTime;
       }
}
```

Like in the *CharacterMovement* script, the *Start* function will get the *NavMeshAgent* component attached to the game object and store it in a variable.

The *Update* function will take the position of the *follow* transform and pass this position to the *NavMeshAgent* variable's *SetDestination* function. This will move the game object to that location.

But we only want to do that sometimes – say, every second – because calculating the path to a location on our nav mesh can be expensive. So the *timer* variable here will ensure that the *SetDestination* function is only called once per second.

Return to the Untiy editor.

You may have noticed that the *FollowPlayer* script has the public *follow* variable. This will be used to hold the game object we want the enemy to follow. Drag your Player object into the field for this variable.

Now you can run your game. You should be able to move your player and bump into obstacles, and the enemy should follow you.