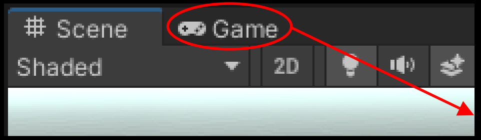
Unity Demo

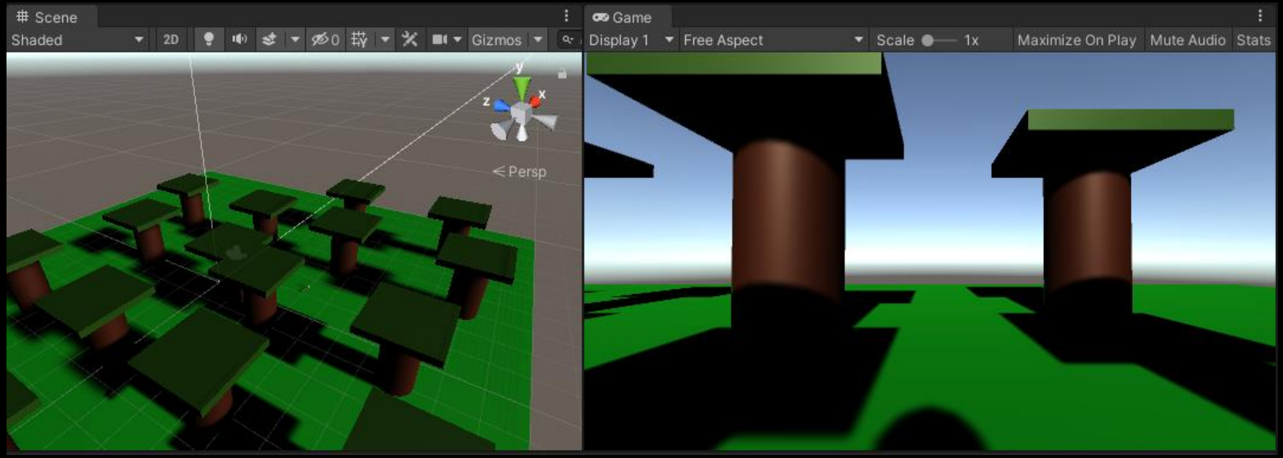
Introduction to the Basics and Unity’s Physics system

Customising the UI

- You can click and drag portions of Unity into separate areas and customise it in a way that best suits your own work environment
- I personally like having the scene view and game view side by side, so that it's easy to click on things and view them in the inspector during runtime

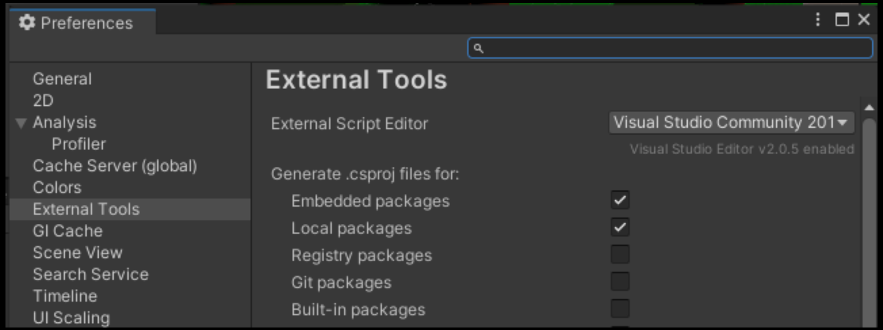


Click the Game tab and drag to the right hand side of the screen to get a view like the one shown on the right



Choosing a Preferred Editor

- You can change the default editor that Unity opens to provide a suitable development environment to use.
- It’s worth noting that by default this will be Visual Studio which has a lot of support with Unity, but there are others that also support Unity that you may prefer to use



External Script Editor: Visual Studio Community 201

Visual Studio Editor v2.0.5 enabled

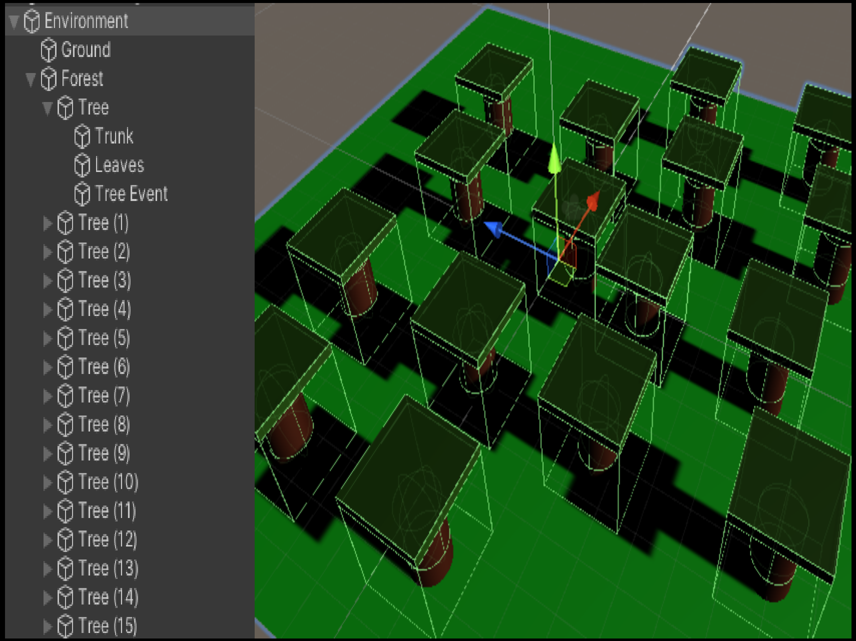
Generate .csproj files for:

Embedded packages	✓
Local packages	✓
Registry packages	
Git packages	
Built-in packages	

To open this window, click:
“Edit → Preferences → External Tools”
Then click the External Script Editor and browse to where it’s “.exe” file is located to change it

Adding a New game object

- A game object is the base class for all entities in a Unity scene. This come in different forms such as: 3D Objects, Lights, Effects, UI Elements and so on.
- There is a Game Object tab where you can add all forms of game objects
- You can also add empty game objects, where you only have a “Transform” component that you controle.g. position, rotation and scale. This is useful for when you want to group a set of game objects together and control them as one. For example, consider the example below:

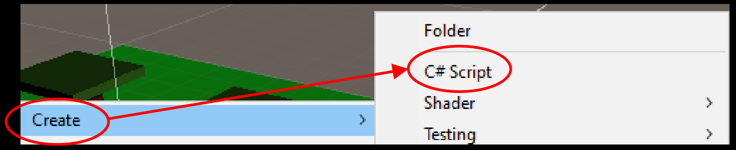


- Here we have an empty game object called the “Environment”, this has 2 children
  - The Ground, which is a 3D Plane game object
  - The Forest, which is another empty game object with another 16 children
    - Each child is a tree, this is an “empty game object” in the sense it doesn’t have a mesh. However each tree also has a rigidbody component added (more on this later). The tree then has 3 children:
      - A 3D Cylinder for the Trunk
      - A 3D Cube for the Leaves
      - An “Empty game object” with a box collider for a trigger event that chops down the tree based on certain conditions

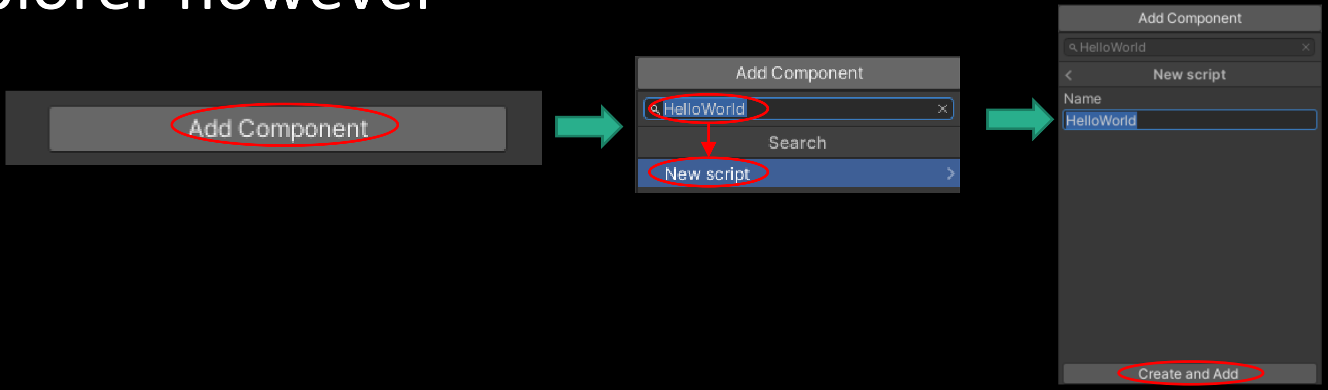
In Summary; empty game objects give you a way to structure your scene so that you can logically group other game objects, without having to move/scale them individually

Creating a New Script

- Navigate to where you want the script to be located in the project explorer and right click. Go to “Create → C# Script” to create a script in that location. You then need to drag this script onto the game object of interest
- Alternatively, to add it directly to a game object, click on the game object in the Hierarchy and click the “Add Component” button in the Inspector tab. Type of the name of the script you want to add then press “New Script”, then “Create And Add”. You may need to relocate the script in the project explorer however

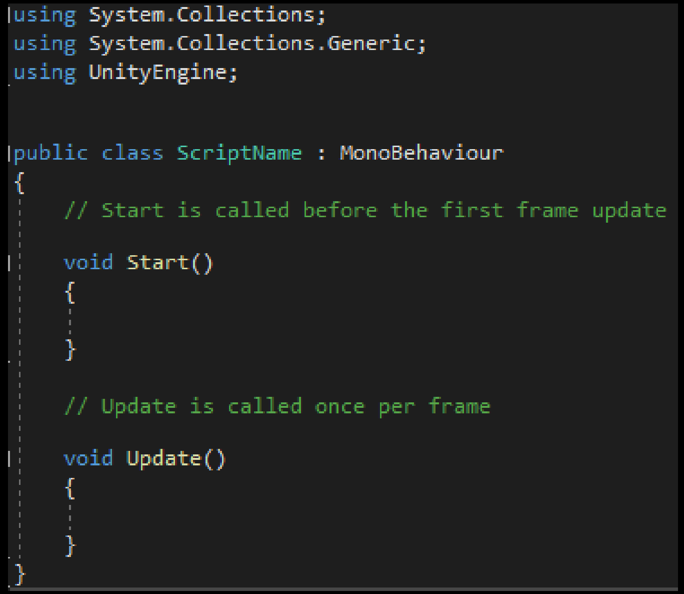


OR



New Scripts

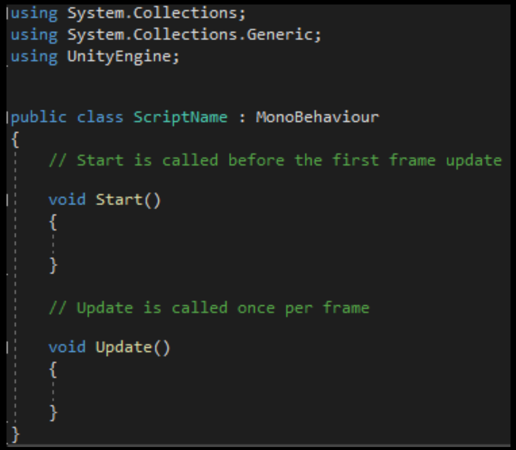
- When creating a new script, by default it will look like the image below
- The three libraries at the top provide basic functions that you can use to work with Unity. The “using UnityEngine” library is probably the most important as it allows you to interact with Unity and many of it’s components, so it’s rare you will ever want get rid of this.
- MonoBehaviour is the base class from which every Unity script derives. When you use C#, you must explicitly derive from MonoBehaviour.



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


public class ScriptName : MonoBehaviour
{
    // Start is called before the first frame update

    void Start()
    {

    }

    // Update is called once per frame

    void Update()
    {

    }
}
```

Start() and Update()

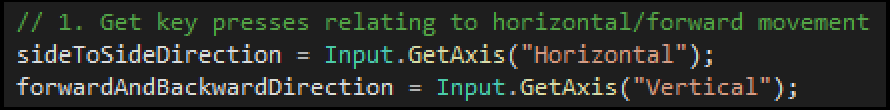
- Looking at the script again, we see we get provided with 2 default functions, Start() and Update()
- The Start Function is called before the first frame. Here you will typically retrieved certain variables and cache them avoid have to repeatedly search for them in Code. For efficiency purposes, it's useful to cache as much as possible
- Update is called every frame, so it's important to make code here efficient. If you have something that requires input, you'll likely need to run code in Update.
    - If you have something that uses physics based functions, and has no code searching for input, that it would be better to swap this for FixedUpdate() which runs at a fixed framerate and ensures the code does not run out of sync with the physics system.



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

public class ScriptName : MonoBehaviour
{
    // Start is called before the first frame update

    void Start()
    {

    }

    // Update is called once per frame

    void Update()
    {

    }
}
```

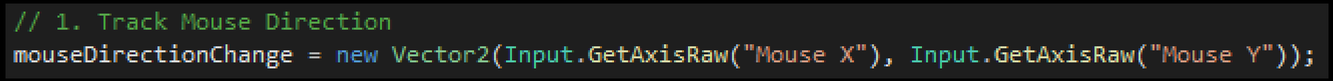
Input Interface

- Unity provide an “Input” interface which allows you to call lots of useful functions such as:
  - “GetAxis” which finds input across a particular axis e.g. Horizontal Input like: “A/D, ←/→”



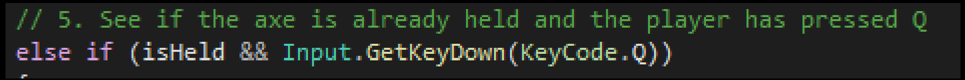
```
// 1. Get key presses relating to horizontal/forward movement
sideToSideDirection = Input.GetAxis("Horizontal");
forwardAndBackwardDirection = Input.GetAxis("Vertical");
```

  - “GetAxisRaw”, as above, but no smoothing applied



```
// 1. Track Mouse Direction
mouseDirectionChange = new Vector2(Input.GetAxisRaw("Mouse X"), Input.GetAxisRaw("Mouse Y"));
```

  - “GetKeyDown”, looks if for a key press of a certain key code

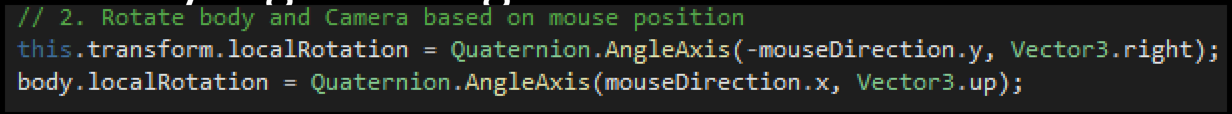


```
// 5. See if the axe is already held and the player has pressed Q
else if (isHeld && Input.GetKeyDown(KeyCode.Q))
```

  - “GetKeyUp”, looks for a key release of a certain key code
  - “GetKey”, looks if a certain key is being held down for a longer period than a single “press

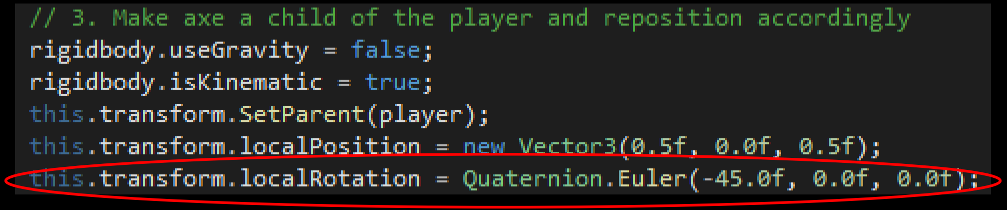
Quaternions

- Quaternions handle rotation in 3D space. They’re 4D vectors and are quite unintuitive to think about, but Unity provide useful functions to help transpose the more intuitive 3D “Euler” angles into these Quaternions, such as:
  - AngleAxis: Rotate by a given angle around a certain 3D axis



```
// 2. Rotate body and Camera based on mouse position
this.transform.localRotation = Quaternion.AngleAxis(-mouseDirection.y, Vector3.right);
body.localRotation = Quaternion.AngleAxis(mouseDirection.x, Vector3.up);
```

  - Euler: Convert a 3D Euler vector into a Quarternion:



```
// 3. Make axe a child of the player and reposition accordingly
rigidbody.useGravity = false;
rigidbody.isKinematic = true;
this.transform.SetParent(player);
this.transform.localPosition = new Vector3(0.5f, 0.0f, 0.5f);
this.transform.localRotation = Quaternion.Euler(-45.0f, 0.0f, 0.0f);
```

Colliders

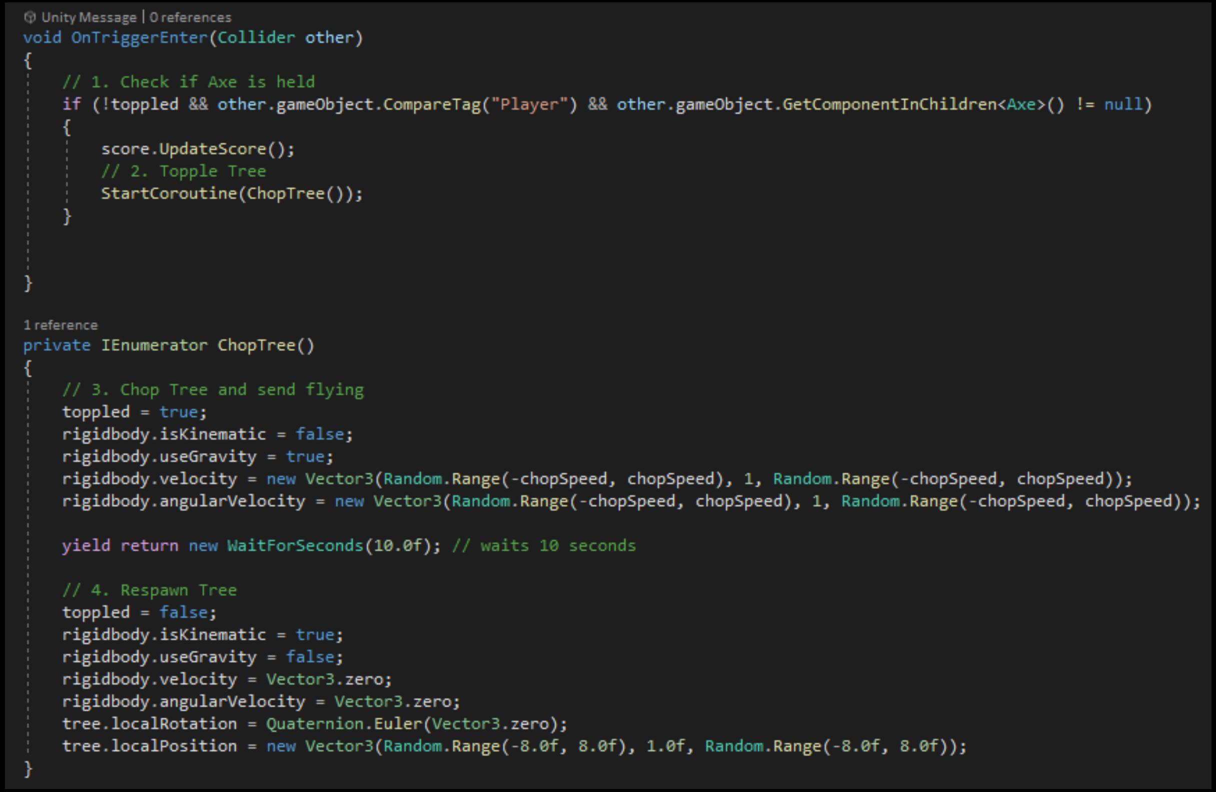
- Colliders are game object components in Unity that help detect if a collision has occurred between two game objects
- They come in different shapes such as primitive: box, sphere and capsule colliders or even a custom mesh collider
    - NOTE: For a mesh collider you will want to produce a low poly version of the model to act as the collider. However, this is still expensive, and so for the sake of optimisation it’s best to use primitive colliders where possible
- A collider can be made a trigger collider so that it only detects logical collisions. E.g. it will state “A collision has been made with this collider, but I’m not going to do anything about it in terms of physics”
- Colliders support functions “OnCollisionEnter” and “OnCollisionExit”, which will run specific code based on if an object has entered the collider radius, or exited the collider.
    - For trigger colliders, you will have “OnTriggerEnter” and “OnTriggerExit” instead.

Rigidbody

- A Rigidbody enables physics simulations on a game object, such as drag, velocity, angular velocity etc.
- Adding a Rigidbody component to an object will put its motion under the control of Unity's physics engine. For example, without adding any code, a Rigidbody object will be pulled downward by gravity and will react to collisions with incoming objects if a collider component is present
- In a script, the FixedUpdate function, mentioned earlier, is recommended as the place to apply forces and change Rigidbody settings (as opposed to Update, which is used for most other frame update tasks). The reason for this is that physics updates are carried out in measured time steps that don't coincide with the frame update. FixedUpdate is called immediately before each physics update and so any changes made there will be processed directly.

IEnumerator

- The IEnumerator class starts a coroutine. A coroutine is can pause during execution, allowing for timed events. For example, the code below will chop down a tree, wait ten seconds, then respawn the tree in a random location:



```
Unity Message | 0 references
void OnTriggerEnter(Collider other)
{
    // 1. Check if Axe is held
    if (!toppled && other.gameObject.CompareTag("Player") && other.gameObject.GetComponentInChildren<Axe>() != null)
    {
        score.UpdateScore();
        // 2. Topple Tree
        StartCoroutine(ChopTree());
    }


}

1 reference
private IEnumerator ChopTree()
{
    // 3. Chop Tree and send flying
    toppled = true;
    rigidbody.isKinematic = false;
    rigidbody.useGravity = true;
    rigidbody.velocity = new Vector3(Random.Range(-chopSpeed, chopSpeed), 1, Random.Range(-chopSpeed, chopSpeed));
    rigidbody.angularVelocity = new Vector3(Random.Range(-chopSpeed, chopSpeed), 1, Random.Range(-chopSpeed, chopSpeed));

    yield return new WaitForSeconds(10.0f); // waits 10 seconds

    // 4. Respawn Tree
    toppled = false;
    rigidbody.isKinematic = true;
    rigidbody.useGravity = false;
    rigidbody.velocity = Vector3.zero;
    rigidbody.angularVelocity = Vector3.zero;
    tree.localRotation = Quaternion.Euler(Vector3.zero);
    tree.localPosition = new Vector3(Random.Range(-8.0f, 8.0f), 1.0f, Random.Range(-8.0f, 8.0f));
}
```

Directing to Unity Sources

- GameObject: https://docs.unity3d.com/ScriptReference/GameObject.html
- MonoBehaviour: https://docs.unity3d.com/ScriptReference/MonoBehaviour.html
  - Start: https://docs.unity3d.com/ScriptReference/MonoBehaviour.Start.html
  - Update: https://docs.unity3d.com/ScriptReference/MonoBehaviour.Update.html
- Input: https://docs.unity3d.com/ScriptReference/Input.html
- Quaternions: https://docs.unity3d.com/ScriptReference/Quaternion.html
- Collider: https://docs.unity3d.com/ScriptReference/Collider.html
- Rigidbody: https://docs.unity3d.com/ScriptReference/Rigidbody.html
- IEnumerator: https://docs.unity3d.com/ScriptReference/MonoBehaviour.StartCoroutine.html
  - Coroutine: https://docs.unity3d.com/ScriptReference/Coroutine.html

Helpful Link for Optimisation Tips

- CGCookie Optimisation Tips: https://cgcookie.com/articles/maximizing-your-unity-games-performance