TUTORIAL THREE: SHATTERING A CUBE

Open a new unity 3D project. Once open, right-click in hierarchy and go to 3D object > Plane. Rename your plane to 'Floor' and scale it out. *Optional: To be more precise, I choose to scale within the inspector > scale and I set each vector to 100. This also takes the plane out to the horizon which I find pleasing to look at. You can also give it a different colour by right-clicking in the asset section and going create > material. Drag and drop this onto your floor either via hierarchy or inside your scene and use the inspector to choose a shade of your liking.*

With your floor all ready, you can now create a cube; by right-clicking in hierarchy, 3D object > cube. Your cube should spawn half inside the floor and so make sure the cube is selected and lift it slightly above the ground, so that later we'll be able to press play and the cube will fall to the floor to shatter. If you'd prefer more precision, in the inspector > transform > position, you can set the Y axis to something like 3.5.
With the cube selected, set your attention on the inventory and go to 'add component' and select Rigidbody to enable physics. Now select the floor and we're going to add component, but this time a box collider. If you look at the box collider component, you'll see a check box for 'Is Trigger'. Since we'll be using OnTrigger, we'll check this box so that Is Trigger is activated.

We can now create our C# script: right click in the assets section, create > C# script, and we'll rename it to 'Explosion'. Drag and drop the script onto your cube in either the hierarchy or within your scene and then double-click it in the assets to have it opened.
We first need to create an 'OnTriggerEvent' and an explode function, setting the floor as the trigger. So below our start and update, we'll create a new private void.

```
private void OnTriggerEnter(Collider other)
{
	If (other.gameObject.name == "Floor")
	{
		explode();
	}
}
```

The explode function will now be called when our cube hits the floor. Make sure that whenever you open a bracket, that you remember to close it. Additionally, it's important to make sure that you're entering the exact name of your collision object as anything other than, in this example Floor, unity will not get understand and so you'll have no result.
To test if this is working or not we'll add in beneath this:

```
public void explode()
	{
```

```
        gameObject.SetActive(false);
    }
```

Now when our cube meets the floor, it'll disappear. You can save your script and open unity to press play and test.

For our explode function, we want our cube to break up into little cubes and we also want them to simulate an explosion with their movement. We can begin this by using 'GameObject.CreatePrimitive', which lets us create primitive objects such as cubes and spheres. We'll also begin to set its positions and give them rigidbody's which will give them mass and let them interact with physics.
Below where we've created our explode function, we'll create a void section as follows;

```
    void createPiece() {


        GameObject piece;
        piece = GameObject.CreatePrimitive(PrimitiveType.Cube);

        piece.transform.position = transform.position;
        piece.transform.localScale = new Vector3(0.2f,0.2f,0.2f);

        piece.AddComponent<Rigidbody>();
        piece.GetComponent<Rigidbody>().mass = 0.2f;
```

After this, we'll need to call our 'createPiece' function from our explode function so head back up to this section and add createPiece(); so that the section of code now looks as follows:

```
public void explode()
    {
        gameObject.SetActive(false);

        createPiece();
    }
```

Save and go into unity to test your scene. The cube now turns into a smaller cube when it meets the floor which isn't quite what we're after. We need some variables to help us create lots of cubes, opposed to just one. Head up to just beneath the beginning of our script (beneath monobehaviour) and type:

```
    public float cubeSize = 0.2f;
    public int cubesInRow = 5;
```

Now we'll modify our createPiece function for creating positioned cubes. Scroll down to this function and make the following additions:

```
void createPiece(int x, int y, int z) {


        GameObject piece;
        piece = GameObject.CreatePrimitive(PrimitiveType.Cube);
```

```
            piece.transform.position = transform.position + new
Vector3(cubeSize * x, cubeSize * y, cubeSize * z);
            piece.transform.localScale = new Vector3(cubeSize, cubeSize,
cubeSize);

            piece.AddComponent<Rigidbody>();
            piece.GetComponent<Rigidbody>().mass = cubeSize;
```

Note: Make sure to always set constant variables (once assigned a value, the value cannot be changed) in this instance.

We'll introduce three nested loops within our explode function to make our positioned 5x5x5 cubes. With this piece we easily ask unity to create 125 pieces off our original cube. We're also introducing the term 'integers' for the first time. In C# an integer, abbreviated to int, means whole numbers. You'd use int to specify that you want your variable to contain only whole numbers. For example, 3 is an integer (whole number), 3.25 is not. Begin with removing 'createPiece();' and then type:

```
public void explode()
      {
            gameObject.SetActive(false);

            for (int x = 0; x < cubesInRow; x++)
            {
                  for (int y = 0; y < cubesInRow; y++)
                  {
                        for (int z = 0; z < cubesInRow; z++)
                        {
                              createPiece(x, y, z);
                        }

                  }

            }
      }
```

Note: Make sure to double check your open/closed curly brackets in this section as it can be easy to forget one.

Save, head to unity, press play and you'll see that your cube splits, but it jerks and holds all the new pieces together which is not what we want
We can fix this by first creating some variables for pivot points so head back up to the top of our script were we set our variables and beneath or cubeSize and cubesInRow add:

```
float cubesPivotDistance;
Vector3 cubesPivot;
```

We need to set a calculation for our pivot distance and then use the value to create our pivot vector. We'll use our Start function for this as it will be a one-time calculation, so head there and after the open bracket, type:

```
      cubesPivotDistance = cubeSize * cubesInRow / 2;
      cubesPivot = new Vector3(cubesPivotDistance, cubesPivotDistance,
cubesPivotDistance);
```

Now we'll use this to subtract from our position within our createPiece function. Where we set our piece position and scale earlier, add ' – cubesPivot; ' to the end like so:

piece.transform.position = transform.position + new Vector3(cubeSize * x, cubeSize * y, cubeSize * z) – cubesPivot;

And now we'll head to our explode function to create some explosion force in our script. We'll do this by establishing an explosion position and what's called an 'OverlapSphere' radius, which then adds force to all colliders within. This part of our script will read as follows:

```
      Vector3 explosionPos = transform.position;
      Collider[] colliders = Physcis.OverlapSphere(explosionPos,
explosionRadius);
      foreach (Collider hit in colliders)
      {
            Rigidbody rb = hit.GetComponent<Rigidbody>();
            if (rb != null)
            {
                  rb.AddExplosionForce(explosionForce, transform.position,
explosionRadius, explosionUpward);
```

Save your script, it is now complete and should look like this:

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

public class Explosion : MonoBehaviour
{

    public float cubeSize = 0.2f;
    public int cubesInRow = 5;

    float cubesPivotDistance;
    Vector3 cubesPivot;

    public float explosionForce = 50f;
    public float explosionRadius = 4f;
    public float explosionUpward = 0.4f;

    // Use this for initialization
    void Start()
    {


        //calculate pivot distance
        cubesPivotDistance = cubeSize * cubesInRow / 2;
        //use this value to create pivot vector)
        cubesPivot = new Vector3(cubesPivotDistance, cubesPivotDistance, cubesPivotDistance);

    }

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

    }

    private void OnTriggerEnter(Collider other)
    {
        if (other.gameObject.name == "Floor")
        {
            explode();
        }

    }

    public void explode()
    {
        //make object disappear
        gameObject.SetActive(false);

        //loop 3 times to create 5x5x5 pieces in x,y,z coordinates
        for (int x = 0; x < cubesInRow; x++)
        {
            for (int y = 0; y < cubesInRow; y++)
            {
                for (int z = 0; z < cubesInRow; z++)
                {
                    createPiece(x, y, z);
                }
            }
        }

        //get explosion position
        Vector3 explosionPos = transform.position;
        //get colliders in that position and radius
        Collider[] colliders = Physics.OverlapSphere(explosionPos, explosionRadius);
        //add explosion force to all colliders in that overlap sphere
        foreach (Collider hit in colliders)
        {
            //get rigidbody from collider object
            Rigidbody rb = hit.GetComponent<Rigidbody>();
            if (rb != null)
            {
                //add explosion force to this body with given parameters
                rb.AddExplosionForce(explosionForce, transform.position, explosionRadius, explosionUpward);
            }
        }

    }

    void createPiece(int x, int y, int z)
    {

        //create piece
        GameObject piece;
        piece = GameObject.CreatePrimitive(PrimitiveType.Cube);

        //set piece position and scale
        piece.transform.position = transform.position + new Vector3(cubeSize * x, cubeSize * y, cubeSize * z) - cubesPivot;
        piece.transform.localScale = new Vector3(cubeSize, cubeSize, cubeSize);

        //add rigidbody and set mass
        piece.AddComponent<Rigidbody>();
        piece.GetComponent<Rigidbody>().mass = cubeSize;
    }

}
```

Head to unity and press play. Your cube will now explode into pieces on contact with the floor. Note: If you select your cube when not in play, within its inspectors we've set a few parameters that you'll be able to experiment with till you get the explosion look that you're after.