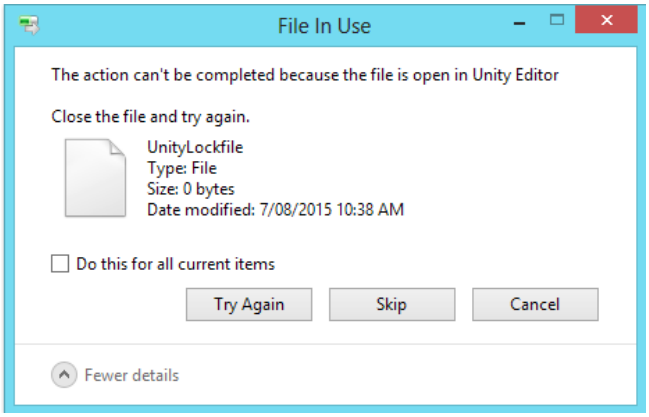
Programming Masterclass for Tank tutorials.

So we all understand how the code is working right now and to overcome a lot of the bugs that have come up this tutorial will be a reference in future for trouble shooting. Alongside this tutorial is a project with errors that have come up so we can work through and fix them all. This will take the pressure off the lectures to enable us to help you students and this will empower you to be able to help yourself.

Table of Contents

Troubleshooting

Locked file while trying to backup your game.

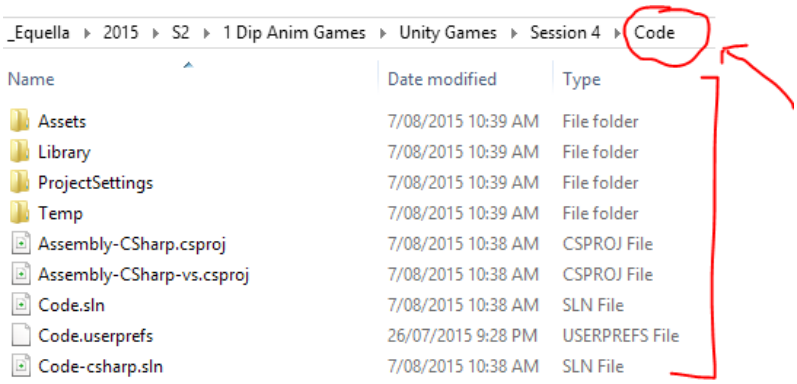


This is because the Temp folder and lock file are being used by Unity.

Solution: Shut down Unity first then copy the entire project folder.

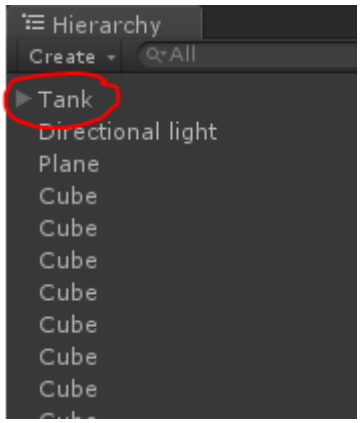
I copied my game (scene file) and I try to load it and there is nothing there.

Solution: You need to copy the entire project folder – the scene files are only a small part of the entire project. See below. Best to go up one folder and copy the entire Code folder in this case.



What does the marking in the tutorial files actually mean?

I use red circle to point out the buttons you should focus your attention on



I use square brackets and squiggle brackets to show you which parts of the code you should add. In this example I mean add all the code where marked next to the squiggly bracket. The code not marked means start writing the code after that code that is already there. So in the example below I'm asking you to add the entire OnCollisionEnter function not the Update function. Add the OnCollisionEnter function below the existing Update function.

```
11      // Update is called once per frame
12      void Update () {
13
14      }
15
16      void OnCollisionEnter(Collision col)
17      {
18          print ("You just hit me");
19          print (col.gameObject.name);
20      }
21
```

What do the Start and Update functions do?

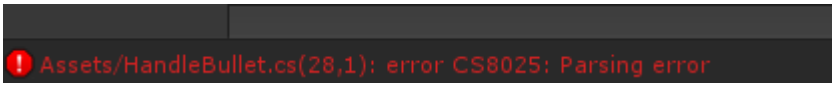
When you create a new script (a class) you automatically get Start and Update in the class file. Start only happens once, the moment the game starts so it's a good place to set initial values for variables. Update happens every CPU cycle – every time the CPU draws to the screen. The Update function is where you want to catch key presses and move things around.

```
1 using UnityEngine;
2 using System.Collections;
3
4 public class CubeCollisions : MonoBehaviour {
5
6     // Use this for initialization
7     void Start () {
8
9     }
10
11    // Update is called once per frame
12    void Update () {
13
14    }
15 }
```

Now were are going to use the broken (scripts with errors) code and run a series of small fixes to make it all run again.

Parse Error.

1. Parse error (end of script reached) Typically you will see this as a red error. It means there is probably brackets / braces missing.



2. Double click on it to open Mono Develop.
3. Carefully check each bracket / brace is in place
4. You can do this by clicking on one brace and Mono Develop shows you its partner. In this case below the Kill has enough braces so that's not the error.

```
13
14      // Update is called once per frame
15      void Update () {
16
17      }
18
19      void Kill()
20      {
21
22          Destroy (gameObject);
23      }
24
25
```

5. In this next case it's also got a partner brace – so that's fine too.

```
13
14      // Update is called once per frame
15      void Update () {
16
17      }
18
19      void Kill()
```

6. The Start method braces are partnered up – so that's not it.

```
// Use this for initialization
void Start () {
    // kill off bullet in 5 seconds
    Invoke ("Kill", timer);
}
```

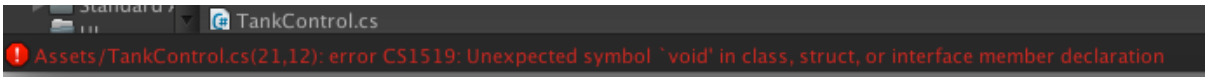
7. But now click on the brace at the top line and notice it doesn't have a partner where is should be down the bottom. There's the error. Add a closing brace to fix the problem. Save and flip back to Unity to fix the problem.

```
public class HandleBullet : MonoBehaviour {

    public float timer = 5;

    // Use this for initialization
    void Start () {
        // kill off bullet in 5 seconds
        Invoke ("Kill", timer);
    }

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

    }

    void Kill()
    {

        Destroy (gameObject);
    }

```

8. Should look like this now if you click on the brace again.

```
public class HandleBullet : MonoBehaviour {

    public float timer = 5;

    // Use this for initialization
    void Start () {
        // kill off bullet in 5 seconds
        Invoke ("Kill", timer);
    }

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

    }

    void Kill()
    {

        Destroy (gameObject);
    }

}
```

Unexpected Symbol (something)…



This means that you probably have missed a semi-colon - the wink character ; at the end of a previous line.

1. To fix the error, double left on the red error.
2. This will open up Mono Develop and take you to the line where the error is.
3. Look up to the previous line to see the error.

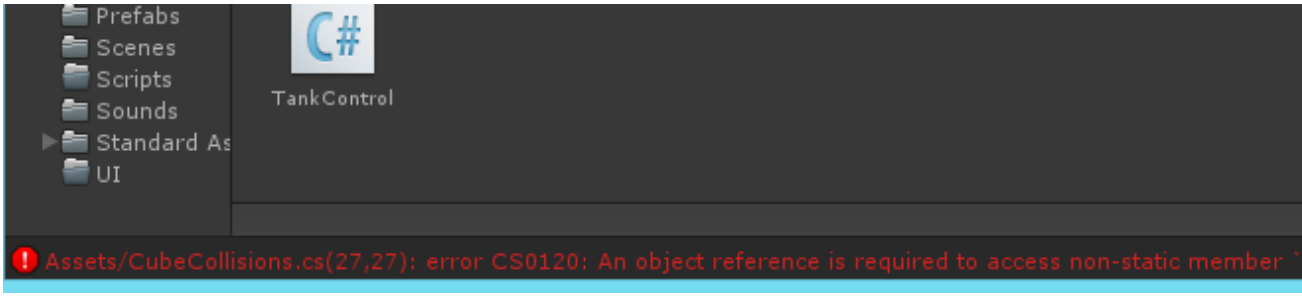
```
15      public GameObject bullet;
16      public Transform muzzle
17
18
19
20      // Use this for initialization
21      void Start () {
22          // set dampen for turning
23          smoothDampenTurn = 1;
```

```
20      // Use this for initialization
21      void Start () {
22          // set dampen for turning
23          smoothDampenTurn = 1;
24          // set slow down for speed
25          smoothDampenSpeed = 1;
26      }
27      void FireBullet()
28      {
29          GameObject obj = (GameObject)Instantiate (bullet, muzzle.position,
30          obj.transform.Rotate (90, 0, 0);
31          obj.rigidbody.AddForce (muzzle.transform.forward * 1000);
32
33      }
```

4. Also notice some other clues you may have seen when the problem occurred.
5. The wavy line under the next line after the missing semi-colon.
6. The red words muzzle.position and muzzle.transform.forward gives you the clue that the error was probably on the same line as the declaration of muzzle further up the script.

Object reference required to access non-static member.

1. You will see this or a similar error down the bottom.



2. Double click on the error. The code looks like this.

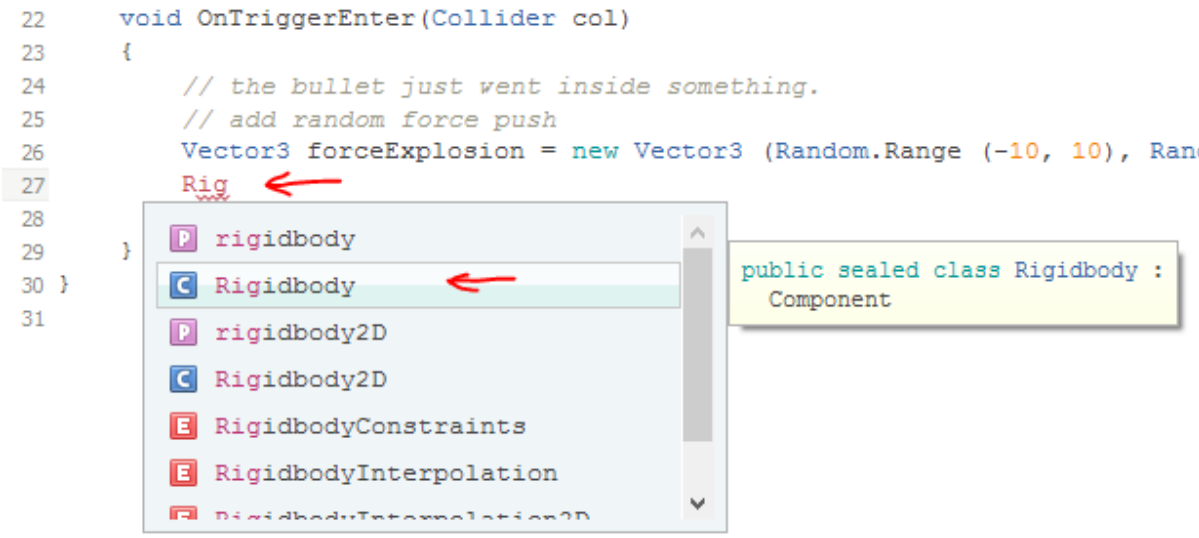
```
21
22      void OnTriggerEnter(Collider col)
23      {
24          // the bullet just went inside something.
25          // add random force push
26          Vector3 forceExplosion = new Vector3 (Random.Range (-1
27          Rigidbody.AddRelativeTorque (forceExplosion);
28          Rigidbody.AddForce (forceExplosion * 100);
29
30          // kill off this bullet
31          Destroy (col.gameObject);
32
33          GameObject.Find ("ExplosionSound").audio.Play ();
34
35      }
```

3. Seems to be okay right? The rest of the error mentioned something about AddForceRelative.

4. So we know it's line 27

```
25          // add random force push
26          Vector3 forceExplosion = new Vector3 (Random.Range
27          Rigidbody.AddRelativeTorque (forceExplosion);
28          Rigidbody.AddForce (forceExplosion * 100);
29
```

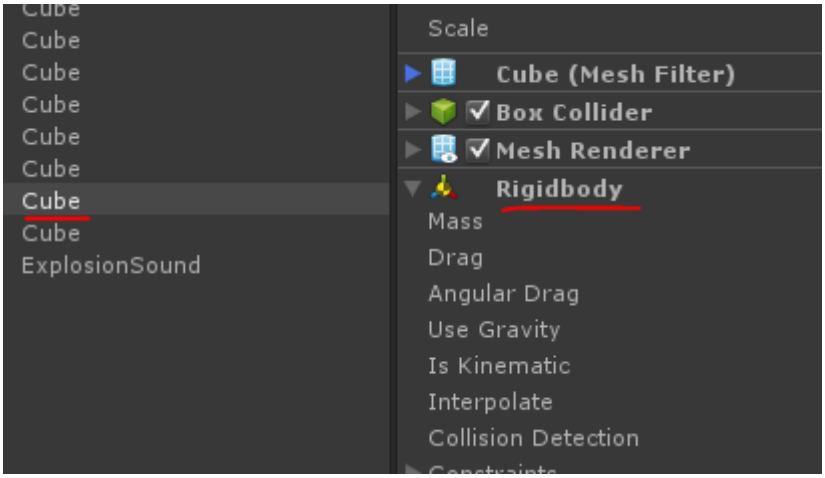
5. It compiled okay. So you would think it was written okay. This problem arose when you were typing and Intellisense (autocomplete) go in the way.



6. It even suggested using the second choice. It did this because typically you would write something like this and essentially it thought that's what you wanted.

```
29
30
31          Rigidbody myRigidbody;
32
```

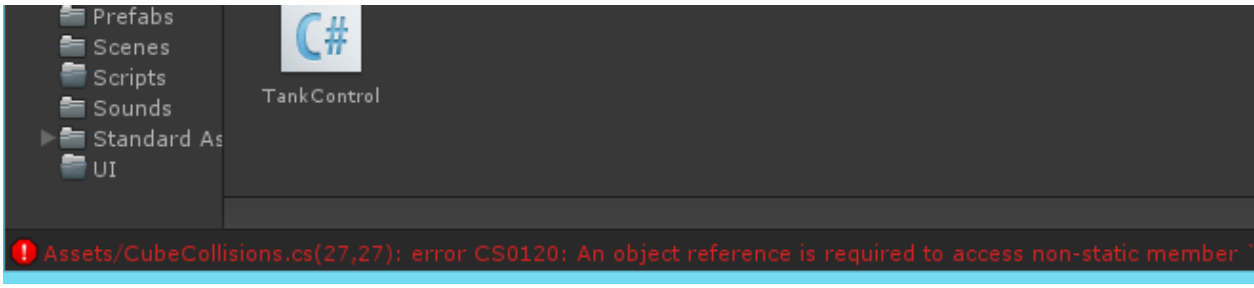
7. You always do this to make a brand new variable…. Data type (Rigidbody) then instance (myRigidbody)
8. But it's wrong in this case because we wanted to perform a function on the rigidbody attached to the object.
9. Go back to Unity and click on a Cube to see what I mean.



10. It confuses the issue because it is also spelt with an uppercase R.
11. You need to know the difference.
12. When you add a Rigidbody to an object you make an instance of the MAJOR Rigidbody. The MAJOR or also known as the BASE CLASS which can create copies of itself – like SOLDIERS.

13. An instance (or reference) of the BASE CLASS (THE MAJOR) is now sitting on the Cube – the SOLDIER – as it's a minion the convention is to set its copy of Rigidbody to lowercase. The SOLDIER references the MAJOR for orders.

14. The next part of the error helps understand this a bit more.



15. **"An object reference"** it says. Ah ha! It needs the SOLDIER not the MAJOR.

16. **"is required to access non-static member"** A member is a function or variable of a BASE CLASS. The MAJOR Rigidbody is known as ***static*** in the universe of C# which means you can't do anything with it other than copy it with all it's functions (orders). And you would want to either. No sense modifying the MAJOR because he knows how to fight. You must give attitude adjustment and training to the SOLDIER (the reference) . As a reference they can be shaped and moulded to become the ultimate killing machine – much like your code.

17. You can't tell the MAJOR what to do but you can give order your SOLDIERS.

18. So the non-static member (function) it's trying to access is AddRelativeTorque which in turn will modify the reference's (SOLDIER) behaviour. In this case we want the rigidbody to get a kick and spin (that's what Torque is – spin)

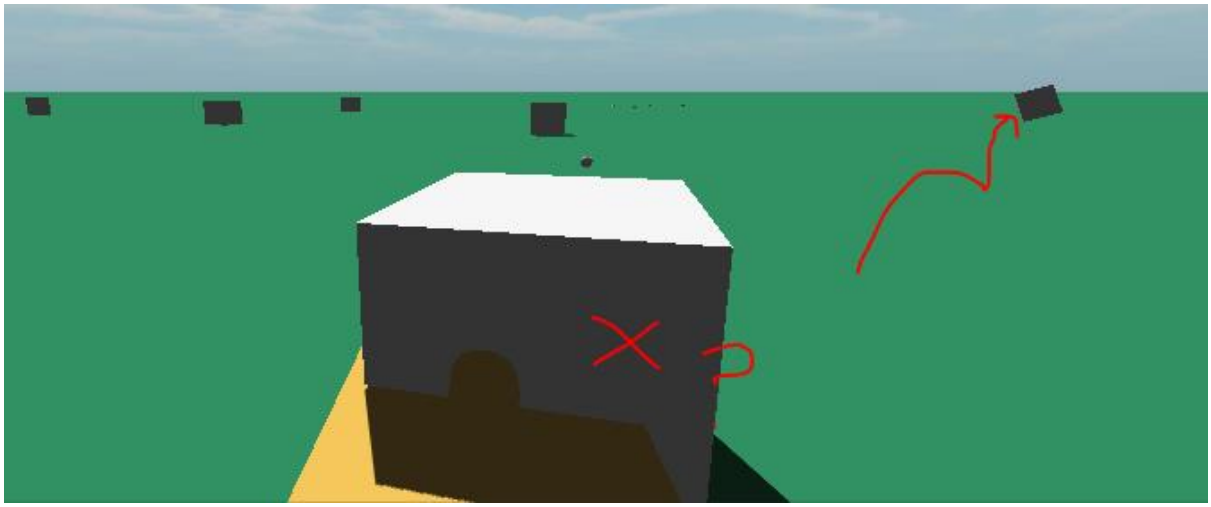
19. So to fix the error we just need to use the reference to the Rigidbody sitting on the cube not the Major (Base Class) … we do it this way… change the Upper case to lowercase – change it for AddForce as well (the following line)

```
22      void OnTriggerEnter(Collider col)
23      {
24          // the bullet just went inside something.
25          // add random force push
26          Vector3 forceExplosion = new Vector3 (Random.Range (-10, 1
27          rigidbody.AddRelativeTorque (forceExplosion);
28          rigidbody.AddForce (forceExplosion * 100);
29
30          // kill off this bullet
31          Destroy (col.gameObject);
32
33          GameObject.Find ("ExplosionSound").audio.Play ();
34
35      }
```

20. Save and flip back to Unity and the error goes away.

My bullets collide with the cubes but my tank doesn't – possibly no console messages either.

1. When you fire the cubes are tumbling around as expected or you're not seeing any messages in the console about cubes.



2. Most probably you've got the OnCollisionEnter function incorrectly capitalised.

```
15
16      void onCollisionEnter(Collision col)
17      {
18          print ("You just hit me");
19          print (col.gameObject.name);
20      }
21
22      void OnTriggerEnter(Collider col)
```

3. Looks fine, compiles without error, but still the problem remains.
4. It's because you wrote a function using correct C# formatting so the compiler thinks it is okay.
5. The problem is that when a collision happens Unity looks for a function in any script called OnCollisionEnter, spelt and capitalised exactly like that. If you change anything about it and the collision occurs you misspelled or miss-capitalised version is NOT called.
6. **Solution** : Capitalise it as required by Unity…

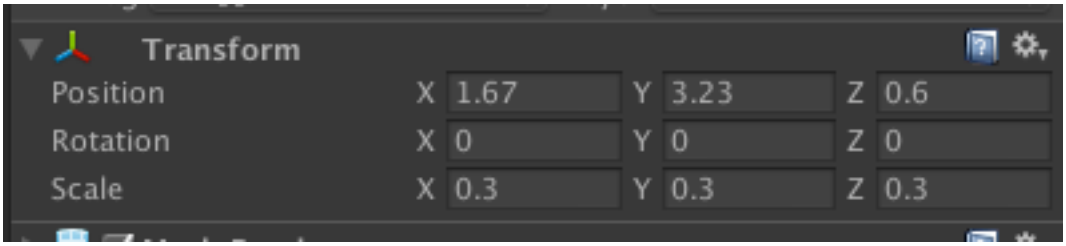
```
15
16      void OnCollisionEnter(Collision col)
17      {
18          print ("You just hit me");
19          print (col.gameObject.name);
20      }
21
```

Understanding the components used so far.

Yeah, you thought that last bit was nerdy. Well now we kick it up a notch.

Most of the code you have used so far is accessing components on the object itself in Unity.

1. In the prefabs folder, click on the bullet object to bring up the Inspector. See the Translate section (or component) at the top

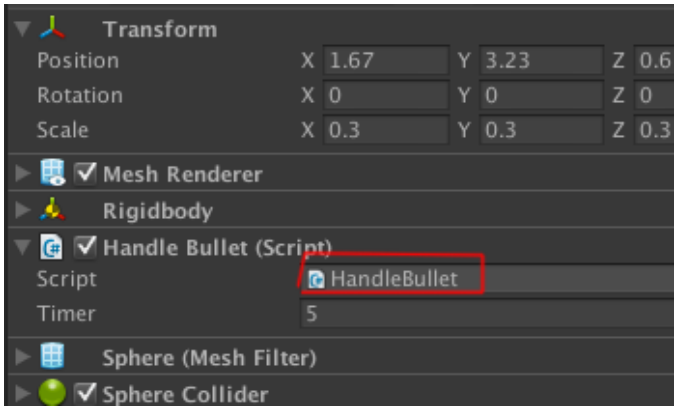


2. It is spelled with a capital T.

3. It has a Position, Rotation and Scale section as well.

Generally, the components and variables in script are lowercase and Unity capitalises them to make them easier to read.

4. So find the HandleBullet script in the list of components in the Inspector of the bullet object and double left click on it to load it up into Mono Develop.



5. If you look at the picture above and see the word Timer and then look at the code below you will see the same word timer is there but its lowercase. Also notice the public float in front of it.

```
public class HandleBullet : MonoBehaviour {

    public float timer = 5;

    // Use this for initialization
    void Start () {
```

a) The reason ***timer*** has changed case is the way Unity shows variables in the Inspector to make it easier to read but it's the same variable.

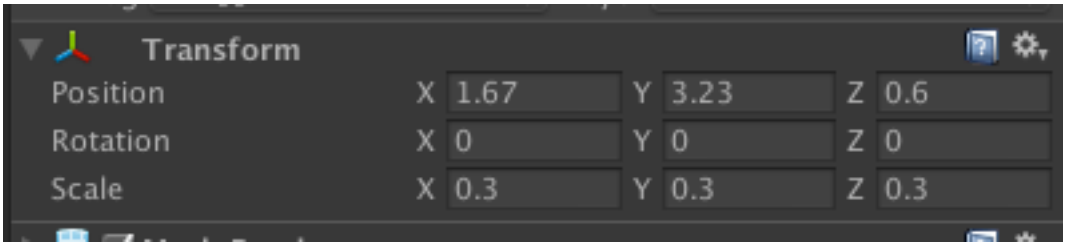
b) The word ***public*** makes it appear in the ***Inspector*** as well so you can change it later. So even though it's 5 in the script you can later change it to another value, say 3. But note that it will still be 5 in the script. So if you make a brand new object that uses the ***HandleBullet*** script it's value ***timer*** will reset to 5 until you change it. Also note that as the ***bullet*** is a ***prefab*** every ***bullet*** you fire will adopt the change to ***timer***. That's what ***prefabs*** do. They are a ***prefabricated*** game object with particular settings that remain the same when created in the game, such as when you fire it (called ***Instantiation*** - the creation of an ***instance*** of an ***object***)

c) The word ***float*** makes ***timer*** an optimised floating point number - or ***float*** for short. Typically maths calculations on computers are slow except for addition and subtraction, and that's using integers (whole numbers). When floating point numbers are used (fractions of numbers using decimal places) the computer has a hard time using them, so the float was introduced giving it a limited number of decimal places, and hence less memory usages and processing time so it was great for 3D games.

6. Look at the **Update** method now. A ***method*** by the way is a another name for a ***function***. A ***function*** is a segment of code surrounded by braces that does something, or performs a function. This ***Update*** method below is just a mock up and would stuff up the bullet so it's just for tutorial purposes. If you're curious about the reason why it would stuff up the bullet is because the bullet also has a ***Rigidbody*** on it which is added to the Rigidbody to project it out of the muzzle. These three lines would upset that so it's not to be used.

a) Look this image below again at the ***Transform*** component, and remember it has **Position**, **Rotation** and **Scale** in it.

```
14      // Update is called once per frame
15      void Update () {
16          transform.position += transform.forward * 10;
17          transform.rotation = new Quaternion (0, 90, 0, 0);
18          transform.localScale = new Vector3 (2, 2, 2);
19          print (Vector3.Normalize(new Vector3(1.2f, 3.5f, 7.5f)).ToString());
20      }
21
```



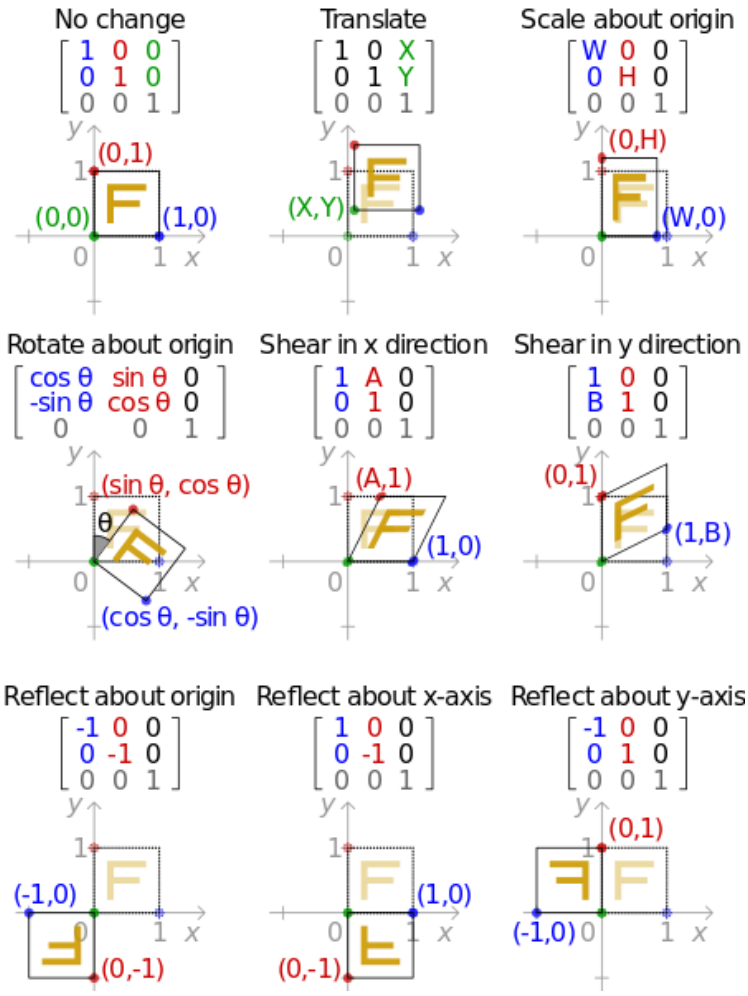
b) Line 16 in the code above modifies the ***position*** variable in the component's ***transform.*** The **+=** symbol is called an ***operator*** - which performs a mathematical operation on a variable. This one is called a ***preincrement*** but just think of it as an **addition**. To understand how to read that line it's adding ***transform.forward * 10*** to ***transform.position***.

c) ***transform/Transform*** is short for ***Transformation Matrix***. Each **matrix** looks a bit different from the others but to move objects around in 3D space they multiply all these ***matrices*** together to end up with a final ***position***, ***rotation*** and ***scale*** of an object - what you end up seeing on the screen.

d) Here's a bunch of practical matrix manipulations.

```
14    // Update is called once per frame
15    void Update () {
16        transform.position += transform.forward * 10;
17        transform.rotation = new Quaternion (0, 90, 0, 0);
18        transform.localScale = new Vector3 (2, 2, 2);
19        print (Vector3.Normalize(new Vector3(1.2f, 3.5f, 7.5f)).ToString());
20    }
21
```

e) Now you know that, you can understand that the ***transform.forward*** is just another matrix added to the ***transform.position***. ***transform.forward*** is a special matrix which contains a direction along **x,y,z** in alignment with the way the object is facing. So adding the **transform.forward** to **transform.position** moves it along the line it's facing direction which could be facing any way. The *** 10** (asterisk 10) means TIMES 10 - you can't use X or x because they are read as variables so the asterisk was used. Another one is the forward slash / which is divide/division because you can't use % as that is a **modulus** operator (essentially used to get the remainder of a calculation) **+** is plus and **-** is take. So ***transform.position += transform.forward * 10*** means move the object along the way it's facing times 10



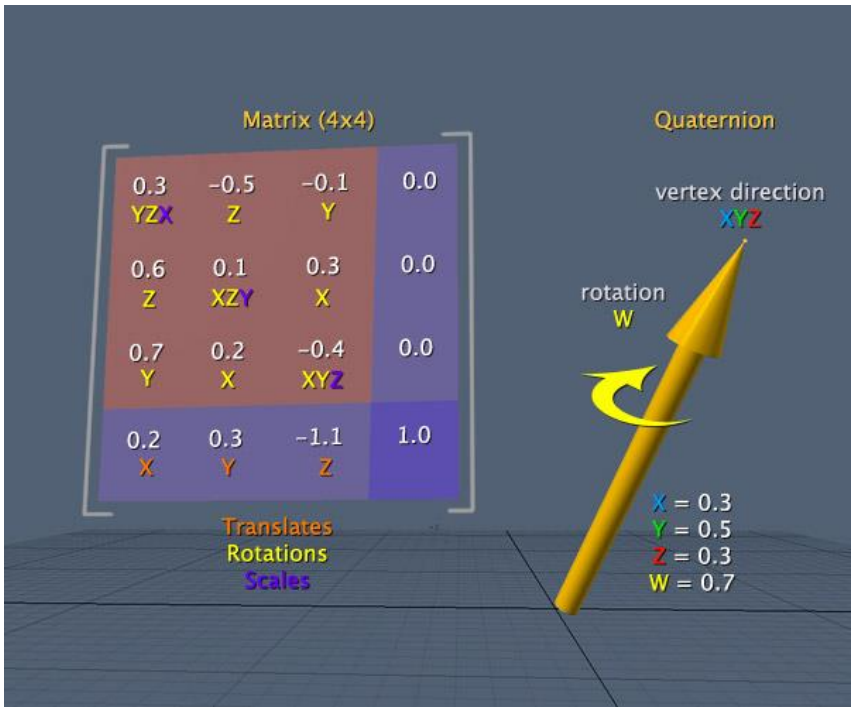
- or move at speed of 10 the way the object is facing. The way its facing is represented as x,y,z and could be something like (-0.5,0.1,0.4) which is called a ***vector***. Most of the time, ***vectors*** like these are ***unit vectors*** which means that none of the values are more than 1 because the size of (or magnitude of) the vector/directional line is 1 in that if you measure the length of the vector/ directional line it's 1 long.

f) Incidentally if you were to use ***Vector3.Normalize(1.2, 3.5, 7.5)*** it would return a new ***unit*** vector (0.1,0.4, 0.9) and it's length (***magnitude*** it one) - so ***Normalize*** creates unit vectors. It's better to have unit vectors because otherwise big number become unmanageable and messy. Also it make thing like setting the movement speed of objects easier to manage.

```
14    // Update is called once per frame
15    void Update () {
16        transform.position += transform.forward * 10;
17        transform.rotation = new Quaternion (0, 90, 0, 0);
18        transform.localScale = new Vector3 (2, 2, 2);
19        print (Vector3.Normalize(new Vector3(1.2f, 3.5f, 7.5f)).ToString());
20    }
21
```

g) Line 17 shows how to rotate an object. It uses assignment operator = (just think of it as "you are now this value") to add a new ***Quaternion*** to the existing ***rotation***. Unity needs any manipulation of a object to use ***Quaternions***.

h) Here's what it looks like.



i) The graph on the left shows you the ***transformation matrix*** for an object which we have seen. The red tinted box covers the **X, Y** and **Z *positions*** of the object (purple text). In the same red tinted area the yellow **X,Y** and **Z** are the angles of ***rotation***. The purple shaded areas handle the object's ***scale***.

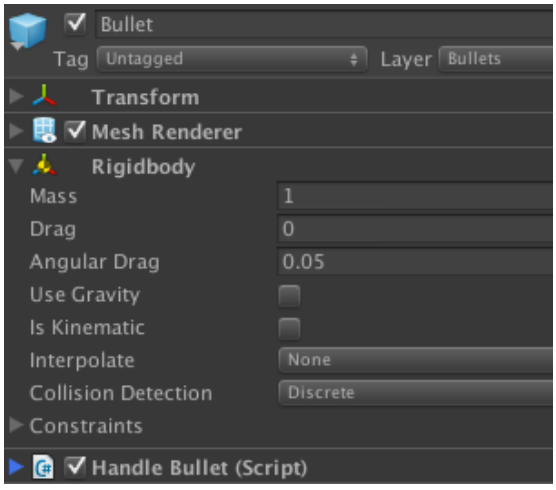
j) A ***quaternion*** has ***x,y,z*** and **w** parts to it. If you look at the big yellow arrow (a vector) to the right of the ***matrix*** you can see it represented graphically. I never spin the **w** coord and always leave it at 0. But the ***x,y,z*** of a ***quaternion*** determine a ***vector*** in 3D space to represent which way to turn the object. Incidentally if you use ***transform.LookAt(someGameObject)*** it will automatically make a ***vector*** for you to point at that object in this way. See how it looks like the arrow is pointing somewhere.

k) A ***vector*** is just a point in space (2D or 3D) and a direction. So in Unity a ***vector*** of (1,0,0) points positive along the ***x axis(left and right)***. A ***vector*** of (0,1,0) points in a positive direction along the ***Y axis (up and down)***. A ***vector*** of (0,0,1) points in a positive direction along the ***Z axis (forward and back)***.

```
14    // Update is called once per frame
15    void Update () {
16        transform.position += transform.forward * 10;
17        transform.rotation = new Quaternion (0, 90, 0, 0);
18        transform.localScale = new Vector3 (2, 2, 2);
19        print (Vector3.Normalize(new Vector3(1.2f, 3.5f, 7.5f)).ToString());
20    }
21
```

l) Looking at line 18 now, the ***scale*** is done using ***localScale*** (just the way it is) and it needs a ***Vector3*** to do it. In this case it scales it by 2 on each axis. It's confusing but just know that to scale an object use localScale.

What is the Rigidbody component?



2. The ***Rigidbody*** is accessed using the lowercase version. Here's an example from the ***CubeCollision*** script.

```
22      void OnTriggerEnter(Collider col)
23      {
24          // the bullet just went inside something.
25          // add random force push
26          Vector3 forceExplosion = new Vector3 (Random.Range (-10
27          rigidbody.AddRelativeTorque (forceExplosion);
28          rigidbody.AddForce (forceExplosion * 100);
29
```

3. The ***rigidbody*** is a special **physics** component that allows objects to bounce, roll, fall, slide, tumble and push other objects around.

4. In the example above you can see in the script it's spelled with a lowercase r but in the ***Inspector*** it's uppercase. You can use lines 27 and 28 on any object in a script and it will compile, but if

you fail to have a rigid body component actually on the object in the Inspector you will get an error like this.

5. Increasing **Mass** on the **Rigidbody** means it required more **force** to push it around but in turn it easily pushes other object around that have less mass than it.

6. **Angular** drag slows the object from rolling.

7. **Drag** slow the object when moving.

8. **Use Gravity** creates gravitational like effects on the object pulling it to the ground.

9. **Is Kinematic** when set to true (or ticked) makes an object stop using physics calculations so the script can move it around but it still does **collisions** as normal on other objects. I typically use it to disable physics until I need the **physics objects** on. As you feel the need to populate your scene with more and more objects containing **rigid bodie**s you may notice the game slows down. To overcome this you could turn off **physics** on the object by ticking **Is Kinematic** until you are ready for them to be affected by **physics**.

10. The rest of the parts of **Rigidbody** we haven't covered yet.