UNIVERSITY OF MALTA
Institute of Digital Games

Creating Virtual Worlds

Lecture 5

Scripts and Components I

Our topics today …

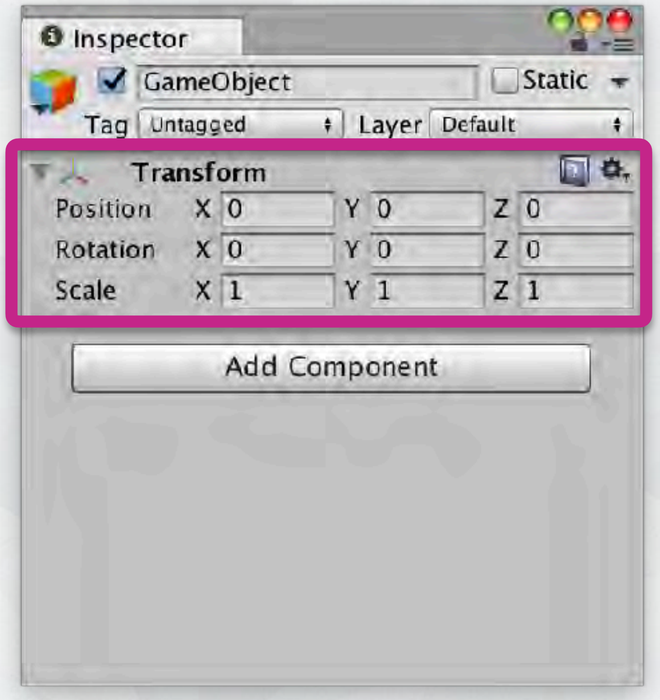
- **Components**
  A look at the most common building blocks
- **Programming Basics**
  Writing instructions to be followed by Unity
- **Data Types and Operators**
  The “vocabulary” of programming
- **Methods (or Functions)**
  Creating re-usable code sections

Components in Unity

We have already used quite a few components in Unity in the past weeks!

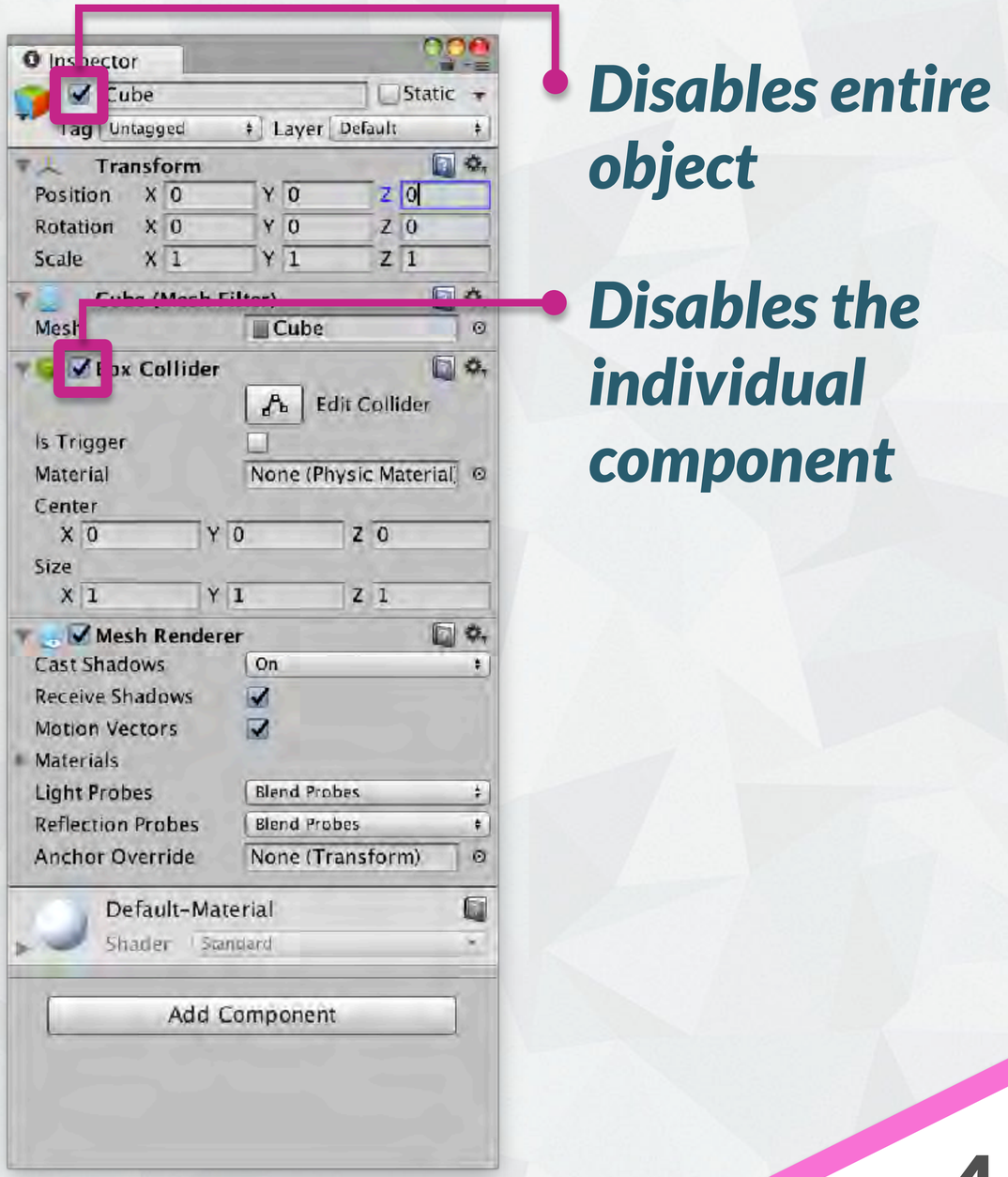
Every object in a Unity scene has, at minimum, a **Transform component** that gives represents its position and orientation in the scene.

… even for objects that don't need one!

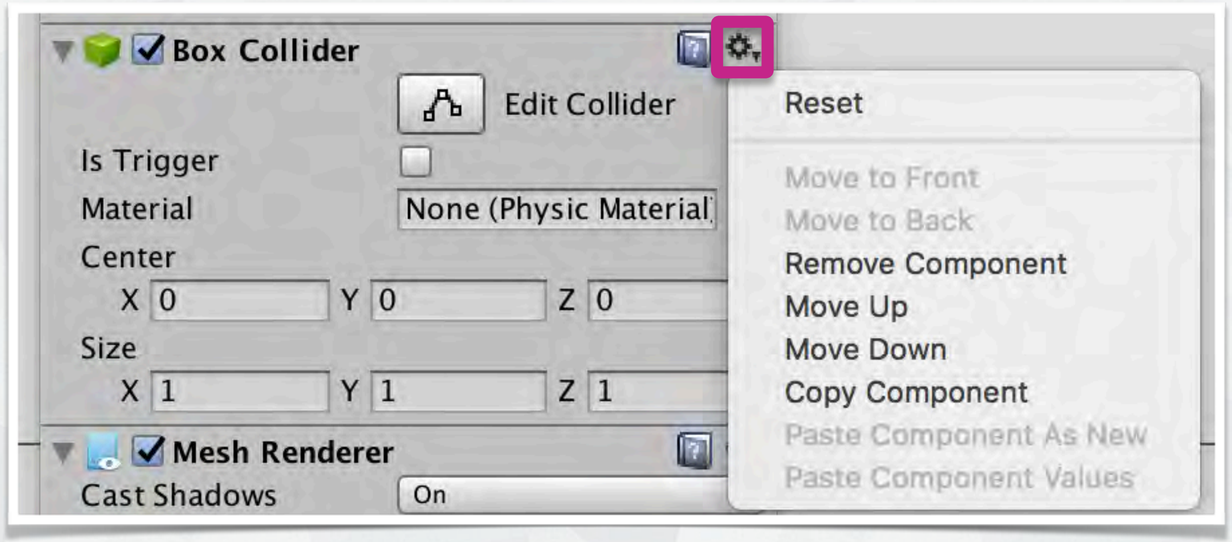


Objects can be deactivated, which also **deactivates all components** that are part of the object.

You can disable individual components — but **other components might depend on them** for their functionality.

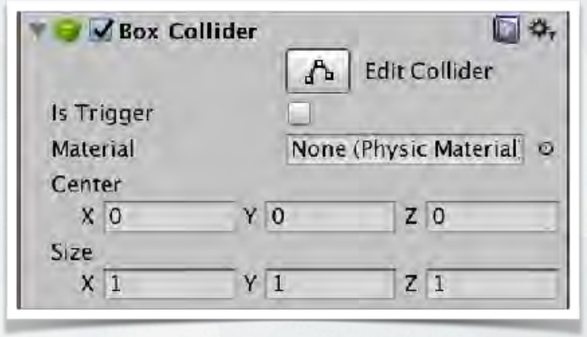


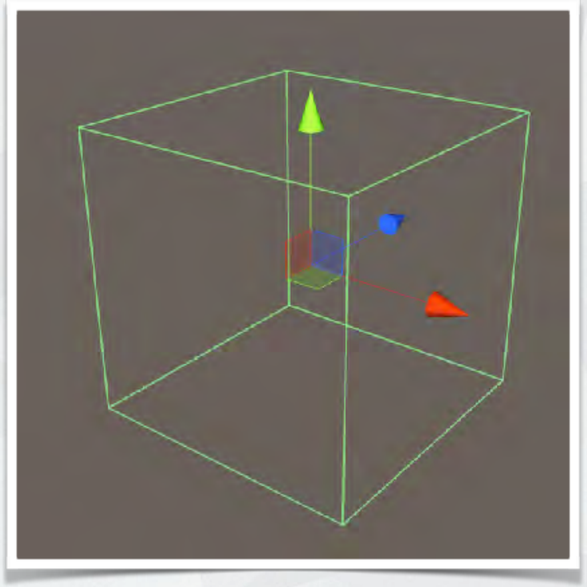
In general, components tend to be **self sufficient**. They incorporate everything that is needed for the functionality they provide.



Components with a dependency can automatically add other components — or trigger a warning on removal.

Important Components





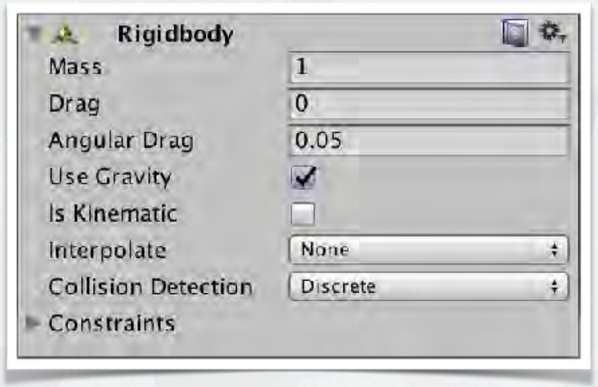
Collider Component

Box, Capsule, and Mesh Colliders

Creates an invisible collision structure for physics calculations. Without colliders, objects can not ‘acknowledge’ each others position in space.

Mesh Renderer / Sprite Renderer

Responsible for displaying a polygonal mesh or sprite to the camera.



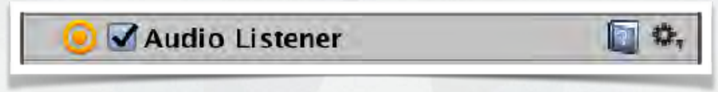
Rigidbody

Controls the position of an object through simulated **physics** (gravity or other forces)

Audio Source

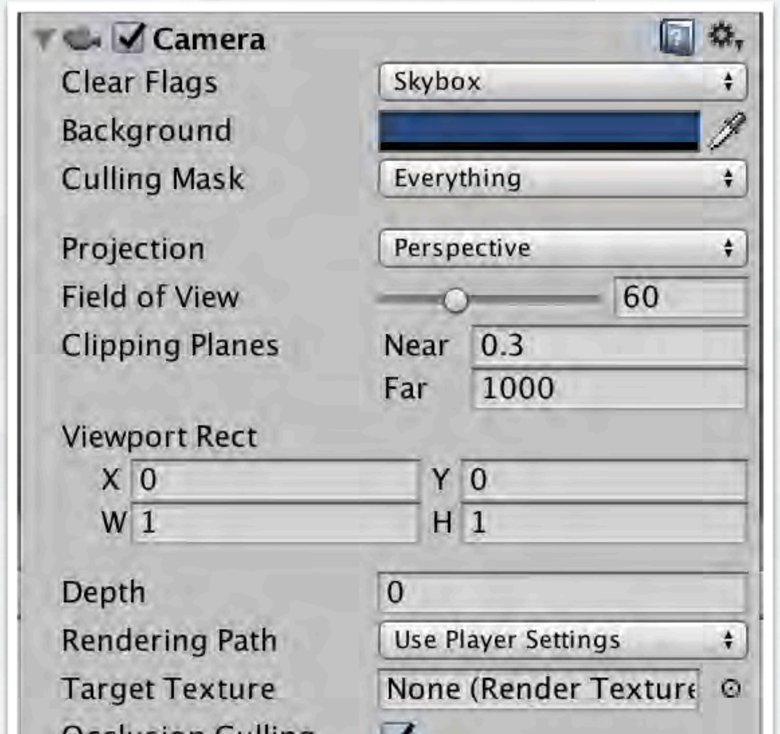
Creates an **audio signal at a given space**

Audio Listener



Usually attached to the main camera, and **determines from where spatial sounds will be heard** (only 1 per scene!)

Not much to adjust …



Camera

Indeed, the **camera is just a component** that can be assigned to any object

Components and Scripts

Every component is, essentially, driven by a script that defines its functionality!

To add functionality, we can write our own scripts that are added to objects just like other components.

Scripts are automatically added to the ‘Add Component’ tab — or can be added via drag-and-drop.

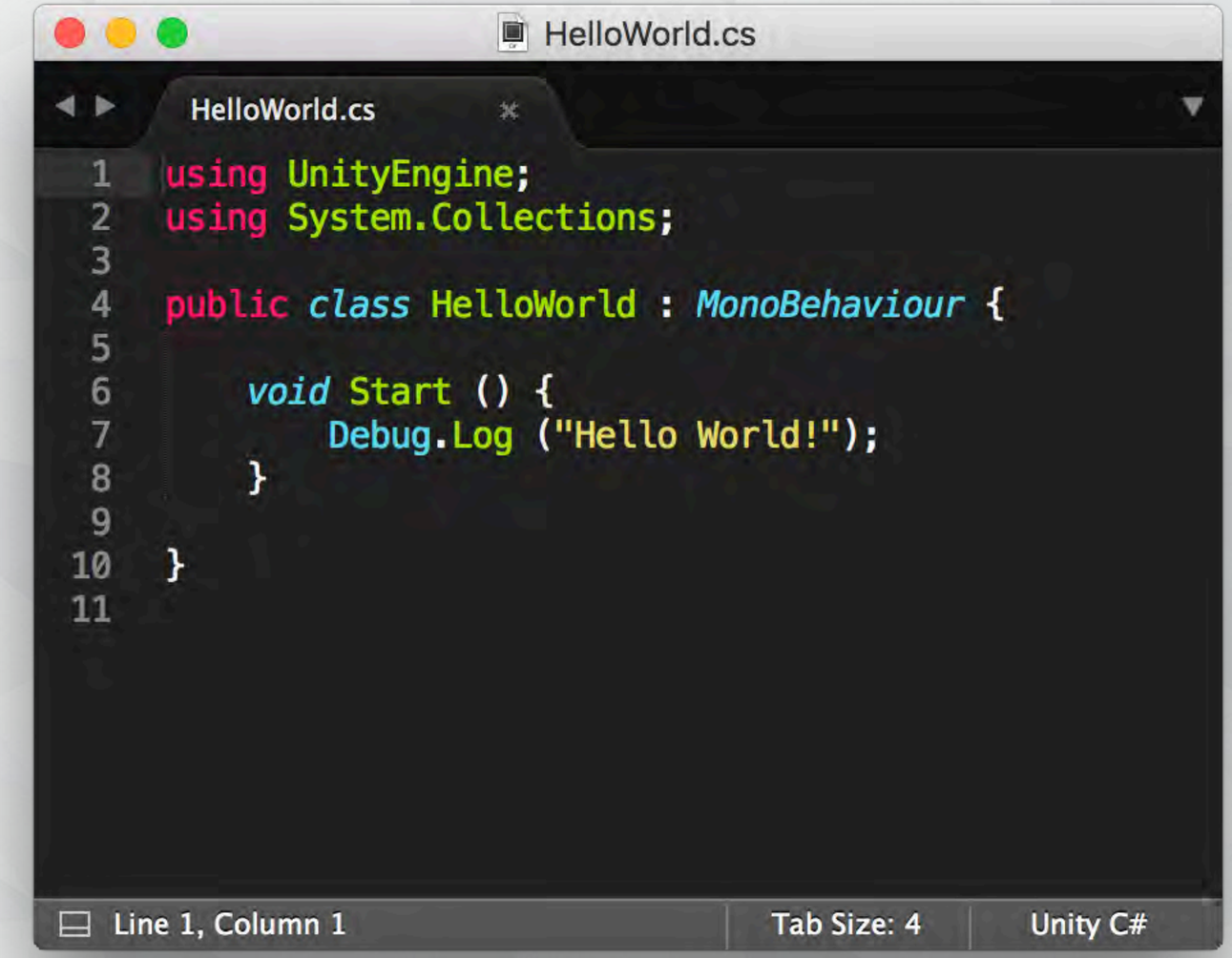
Programming Basics

Programming essentially means to write out a carefully worded list of instructions, using a specific vocabulary and way of writing.

Unity interprets the written code and translates it into instructions that can be passed on to the hardware. Usually, Unity triggers a warning for problematic instructions — at worst, it crashes :D

Scripts in Unity can be written in **UnityScript*** and **C#**

We will be looking at **C#** — but the fundamental logic is very similar!



** Also referred to as JavaScript — but it's not quite the same!*

Structure

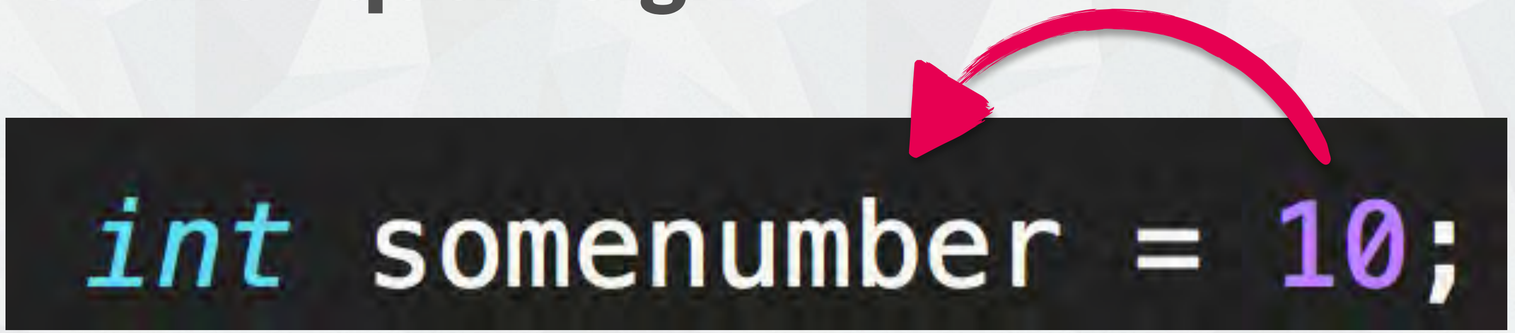
- Instructions are carried out in **sequence**
- Most instructions need to be ended by a **;**

```
public class TestScript : MonoBehaviour {

    int somenumber = 10;
    float anothernumber = 10.4F;
    string sometext = "some text indeed";
    bool someswitch = true;


}
```

Variables and Data Types

Variables are designated ‘holders’ of data. The kind of data it holds is defined when data is passed ‘into’ the variable.

The value that is to be assigned needs to be on the right side of the equals sign!



```
bool isWalking = true;
bool isRunning = false;
```

Booleans

Can be either true or false — often used as a sort of on-off-switch

```
int age = 15;
int cash = -200;
```

Integer

Indicates a whole number, positive or negative

```
float actualAge = 15.52493F;
float actualCash = -200.04F;
```

Float

Indicates a real number that can include a decimal point

```
string name = "Paul";
string id = "4019";
```

Strings

Indicate an array of unicode characters

```
Vector2 origin_2D = new Vector2(0F,0F);
Vector3 origin = new Vector3(0F,0F,0F);
```

Vector2 / Vector3

A Unity data type that holds an array of floats — typically indicating positions in space

Operators

We already saw one operator in use — the designation operator **=**

Without a designation, variables are **initialised**, but either **remain empty** or hold a **default value.**

```
bool hasTriggered; // initialised as hasTriggered = 'false'
int number; // initialised as 0
float anothernumber; // initialised as 0.0F

// Oh yeah: '//' starts a comment - ignored by Unity
```

Arithmetic Operators: + — * /

```
int number;
number = 5 + 5;           // result: 10
number = number + 10      // result: 20
number = number / 3       // result: 6  -- not 6.666 or 7
number = 3 * 5            // result: 15
```

```
float blubb;
blubb = 1.0F;
blubb = number * blubb;          // result: 15.0F
blubb = 0.5F * (2.0F + 10.0F);   // result: 6.0F
```

Relational Operators: == != > < <= >=

These return true or false!

```
int a = 5;
int b = 10;

a == b;      // False
a == b / 2; // True
a < b;       // True
a != b;      // True
```

```
string word = "5";
word == 5                // False
word == "5"              // True
word != 5                // True
```

Relational Operators: == != > < <= >=

These return true or false!

```
int a = 5;
int b = 10;

a == b;      // False
a == b / 2; // True
a < b;       // True
a != b;      // True
```

```
string word = "5";
word == 5                // False
word == "5"              // True
word != 5                // True
```

Sloppy!

Logical Operators: && || !

Combining multiple true / false statements

```
bool jepp = true;
bool nope = false;

!jepp;              // False

jepp && nope;    // False
jepp || nope;    // True
jepp && !nope;   // True

!(5 == 5) && ((jepp == 10) || !nope);    // False
```

If Statements

Trigger parts of the code if a statement is true

```
bool hasTriggered = true;

if (hasTriggered) {
    hasTriggered = false;
}

// result: hasTriggered is false
```

```
if (5 == 10) {
    // Triggered if true
} else {
    // Triggered if false
}

int a = 2;

if (a == 5) {
    // Triggered if a equals 5
} else if (a == 2) {
    // Triggered if a equals 2
} else {
    // Triggered if none were true
}
```

Methods (or Functions)

Methods are parts of code that are executed whenever the method is called. But careful — variables that are initialised 'inside' a method are not available 'outside' of it!

```
void Boo () {
    // Do something
}

Boo();  // calling the method
```

```
bool TrueOrFalse () {
    return true;
}

int GiveMeFive () {
    return 5;
}
```

Methods are defined by the data type that they return, or by **void** if they do not return anything.

```
bool IsBiggerThanTen (int input) {
    if (input > 10) {
        return true;
    } else {
        return false;
    }
}
```

To make methods more reusable, we can define parameters that the method needs to execute!

Break Time!
Any questions?

After the break:

Unity Practice — Continue building the scene