Handout – Scripting

C# Basics:

C# is a modern, fast programming language developed by Microsoft, with similar syntax to C++ and Java. This section explains some of the basic concepts of scripting / programming.

Variables

Variables allow you to store information in memory. There are many different types of variables in C# which are the foundations of every program.

The most common ones that you will use in Unity are:

- Integers - whole numbers
- Floats - decimal numbers
- Strings - words
- Booleans - true or false values

You can declare them like this:

```
type name = value;
```

Some example variables are shown here:

```
//Declare variables

int health = 100;

float speed = 91.5f;

string name = "Bob";

boolisAlive = true;
```

Debug.Log simply outputs the value of the variable to the debug window in Unity.

If Statements

Generally, code is executed from top to bottom. You can use if statements to control your program flow based on certain conditions.

They look like this:

```
if (condition)
{
//If this condition is true
//this code will be executed.
}
elseif(otherCondition) //Optional
{
//If the other conditions(s)
   //were false, this code will be
//executed.
}
else//Optional
{
//If all other condition(s) were
   //false, this code will be
   //executed.
}
```

Else if and else are optional sections you can add onto your if statement, allowing for more control over how your program behaves. The condition is always a bool and will always equal true or false. Note that only one section is ever executed.

Arrays

When working with large numbers of objects, it's not really practical to declare every single variable. It would take a large amount of time and space to declare and use each variable.

```
//Declare arrays of size 5
publicint[] playerScores = newint[5];
publicVector2[] playerPositions = newVector2[5];

//Access individual array item
//Indices range from 0 to 4 for an array of size 5
playerScores[0] = 5;
playerPositions[0].x = 15f;
```

Arrays allow you to store a large amount of items into one variable. Each individual item can be accessed by an index starting at 0 for the first index.

Loops

Loops allow us repeat sections of code as long as a condition is true. This operation can be very handy for arrays or other large sets of data.

There are four different types of loops:

- While
- Do While
- For

The syntax for each of these is listed here:

```
//While loops are straightforward loops that execute as long as the
condition is true
while (condition)
{
//Code
}

//Do while loops are similar to while, but are guaranteed to execute at
least once
do
{
//Code
} while (condition);

//For loops have a different syntax to other loops
//The initializer gives the starting point for a loop
//The code in the for loop will execute as long as the condition is true
//The expression is something you want the loop to do on each iteration
for(initializer; condition; expression)
{
//This code will be executed while the condition is true
}
```

Functions

Functions (methods) allow us to split our method up into smaller, more manageable chunks. As a rule, if you find yourself writing the same block of code more than once, it could probably be used as a function.

You have probably used functions already without realising!

The syntax for declaring a function is as follows:

- The function name is a unique name to give to your function
- The type is the data that you would like your function to return (`void` if there is nothing to return)
- Parameters are data types that are being passed into our function

```
//Parameters are optional
return_typeFunctionName(type1 parameter1, type2 parameter2)
{
returntype;
}
```

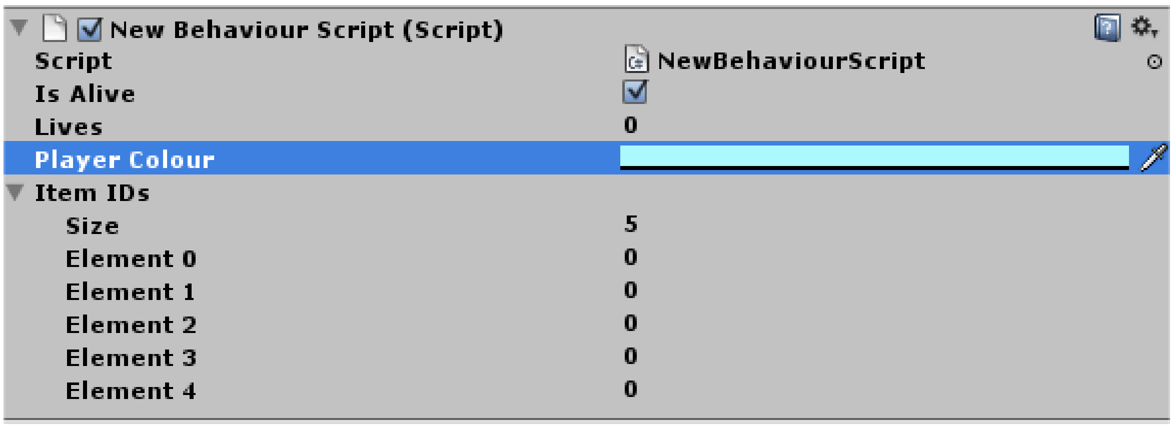
Scripting in Unity:

Unity makes it easy for us to script in C# by providing several built-in methods and properties we can access to control the position, rotation and other components of our Game Objects.

The Inspector

Unity's inspector makes it easy for us to change variables on the fly or to see what information they contain. You can use the keyword`public` in front of the variable to see it in the inspector.

```
//Public variables
publicboolisAlive;
publicint lives;
publicColorplayerColour;
publicint[] itemIDs = newint[5];
```



```
New Behaviour Script (Script)
Script            NewBehaviourScript
Is Alive          ✓
Lives             0
Player Colour
Item IDs
  Size            5
  Element 0       0
  Element 1       0
  Element 2       0
  Element 3       0
  Element 4       0
```

Monobehaviour

Unity also has several built in functions that get called automatically at different times. By default, any new script you create has the `Start()` and`Update()` functions. Some other useful ones are:

- `OnGUI();` - used for GUI events
- `FixedUpdate();` - used for Physics
- `OnCollisionEnter`- called when a collision is detected by physics

For a full list, look here: http://docs.unity3d.com/ScriptReference/MonoBehaviour.html

Components

You can also access various components from a script attached to a game object.

The most common one you will probably use is *transform*and its respective methods. Most component's names correspond to a similar name in the inspector. For example, *Animation* in the inspector is *animation*, *Camera* is *camera*, and so on.

For example, we've seen how we can get the Rigidbody component that is attached to a game object, and set its properties directly:

```
m_Rigidbody = GetComponent<Rigidbody>();
m_Rigidbody.isKinematic = false;
```