Unity Tutorials 8 – Basic Scripting

I – Intro to JavaScript

Object-Oriented Programming
JavaScript is an object-oriented programming language. In programming, an object is a combination of variables and methods. Object-oriented languages are easy to visualize. For example, imagine that a car is an object. A car in turn has a number of objects within it. Cars have four wheel objects. Each wheel object has its own variables such as size and colour. Another object that a car object contains is its doors. Each door has its own variables, but only those that are useful to the program. For example, the car may have a colour variable. This would have a "string" data-type. A string is usually a word or a sentence. The door may also have a child-lock variable that can either be engaged or disengaged. This would be a "boolean" data-type, which can only either be true or false. Let's say the wheels are not objects, but are instead a variable. Let's say that the wheel variable simply determines the number of wheels. This would be an integer data-type, which is a number that is not a fraction. The "wheel" variable for a car would generally have a value of "4".

Variables
A variable is something within an object that holds a value, or information. As mentioned above, there are different data-types that a value can have. The most commonly used in Unity are integers, strings, and booleans. 0 is an example of an integer, zero is an example of a string, and false is an example of a boolean.

Methods
A method is a set of step-by-step instructions that are to be followed. They are used to complete specific tasks, generally by manipulating variables. For example, a car object may have a method called "accelerate". This method would follow a set of steps that eventually end with the wheel objects spinning faster and faster.

Declaring and Defining Variables
In order for a method to use a variable it must first be declared and defined. Declaring a variable basically tells the rest of the program that it exist. Defining a variable is the act of assigning a value to it. If a variable is not defined it will have a value of "null". Look at the JavaScript statement below.

var colour = green;

The first part, "var", is necessary for declaring variables. The second part, "colour" is the name of the variable. Each variable (and method) must have a name that will be *used to refer to it. The "=" means that we are also defining the variable. "green" is the string value of this variable. The ";" means the end of the statement. If the statement were*

```
var colour;
```

then "colour" would be a new variable with a value of "null". If the original statement was in Java form (as opposed to JavaScript) it would look like this:

```
String colour = green;
```

This is because in Java it is also necessary to declare the data-type. JavaScript automatically assigns the most appropriate data-type.

Classes

A class is basically a large method used to create objects. It begins by declaring the object's variables then lists different methods (often know as "functions" when they are within a class) that an object can perform. Consider the statement below.

```
Car vehicle1 = new Car();
```

This is how an object is created within a program. You'll notice that it is much like declaring and defining a variable. The "Car" at the beginning means that vehicle1 is a "Car" object. In order for this to work, there needs to be a class called "Car" somewhere within the program. "vehicle1" is the name of the car. The "=" denotes that we are assigning a value to this Car object. "new Car()" is calling (telling the programming to find and initiate) the method for creating a car object.

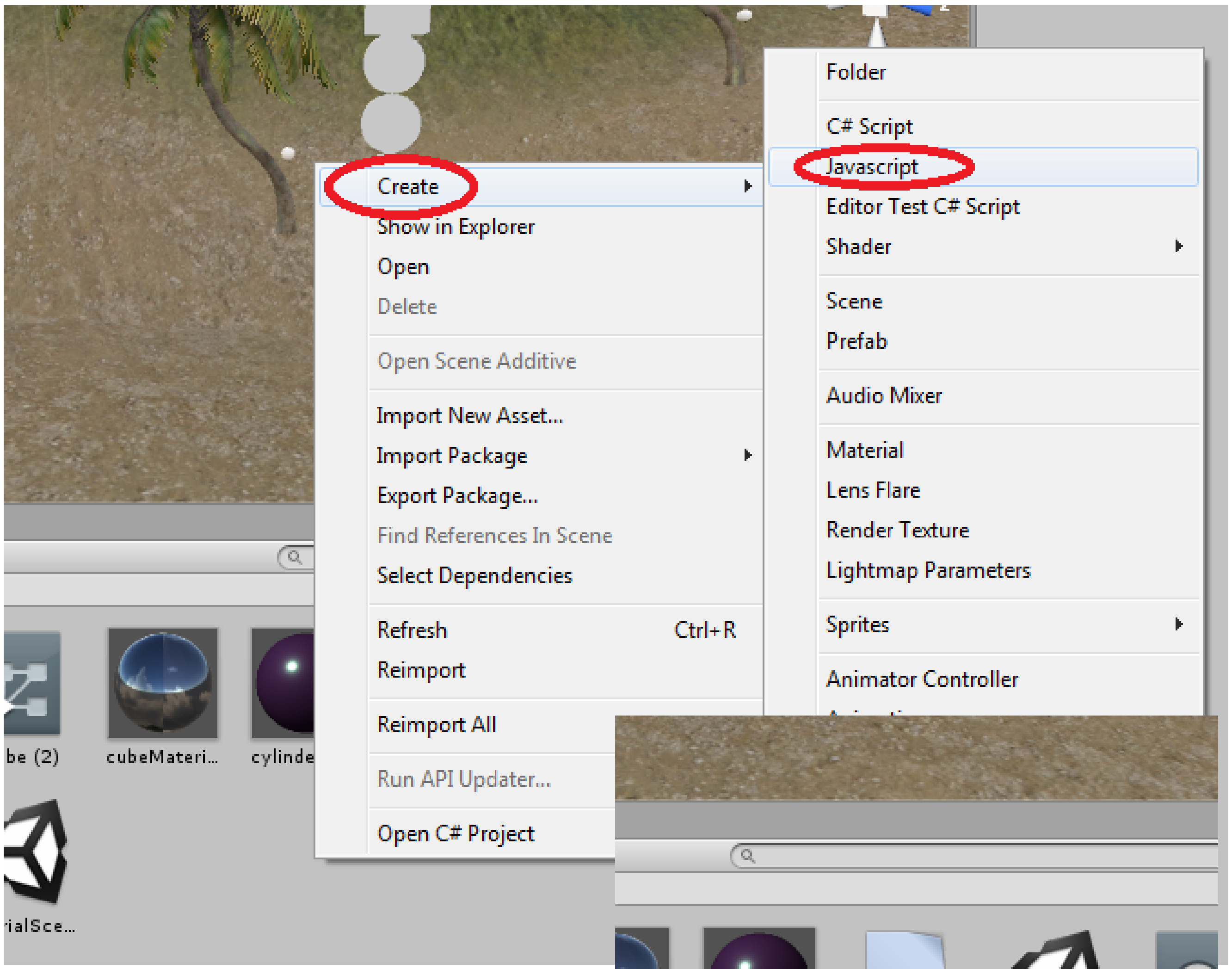
Unity Tutorials 8 – Basic Scripting

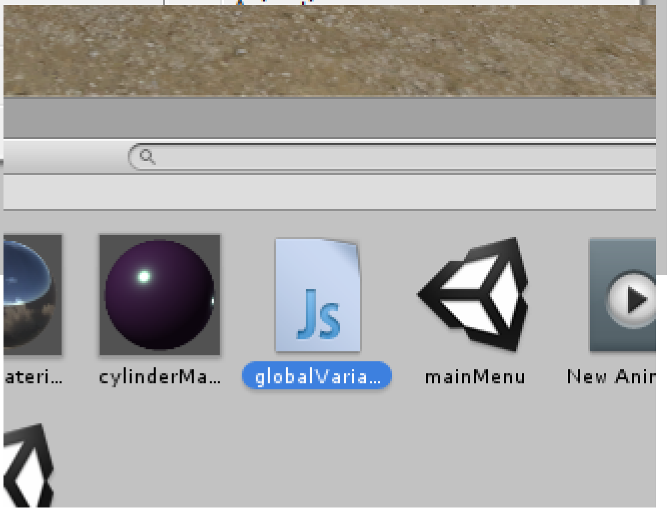
II – Health Bars

Global Variables

A global variable can be accessed by any part of a program. They are generally frowned upon because they can cause trouble if the programmer uses the same name for other variables. We will be using global variables anyway for simplicity's sake.

Right-click in your Assets and go to "Create" and "Javascript". Rename it as "globalVariables".



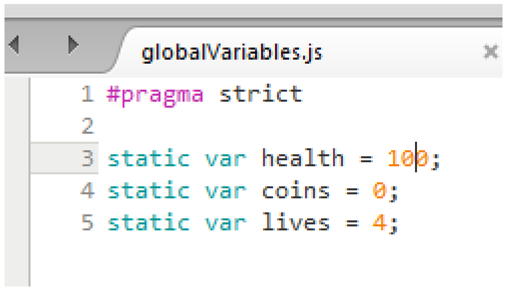


Now double-click on your script to open it in Unity's "MonoDevelop" program.

Unity Tutorials 8 – Basic Scripting

Erase everything except the first line that says “#pragma strict”.

Now type the following:



globalVariables.js

```
#pragma strict

static var health = 100;
static var coins = 0;
static var lives = 4;
```

Technically each statement doesn't need to be on a new line. This is for organization purposes.

Press the CTRL and S keys simultaneously or click on “File” then “Save” to save.

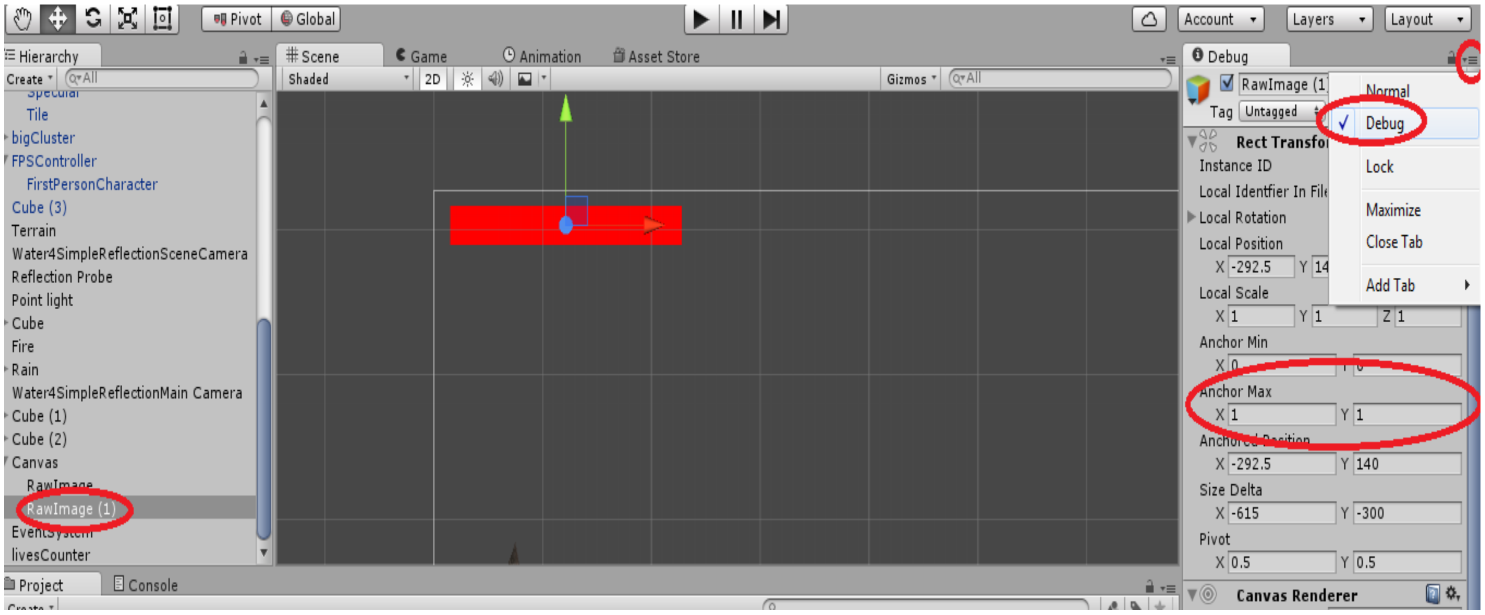
In this script we have declared and defined three variables. In JavaScript the “var” denotes that we are declaring a variable, then we name it, then we assign a starting value, and finally the “;” declares the end of the statement. The “static” at the beginning is required for this variable to be accessed by other scripts. A static variable is one that will last for the entirety of the program.

Assigning Scripts
We are now going to assign a class containing the remainder of our health bar-related methods to the obstacles. Generally we would split them up between different classes in order to better organize our script and minimize redundancy. For example, the enemies could have a script that detects collision then calls the method that takes care of health bar depletion from another class. This way we won't have to have the entire script in each enemy and it would only need to exist once. For simplicity's sake though we will not be doing this.

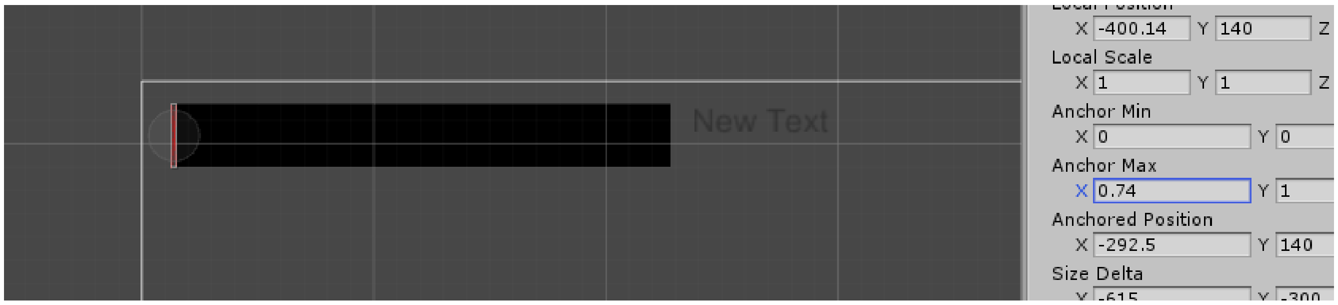
Create a new Javascript code and name it “obstacleCollision”. Double-click it.

Before doing anything, go to your RawImage (1) and enter “Debug” mode in the inspector.

Unity Tutorials 8 – Basic Scripting



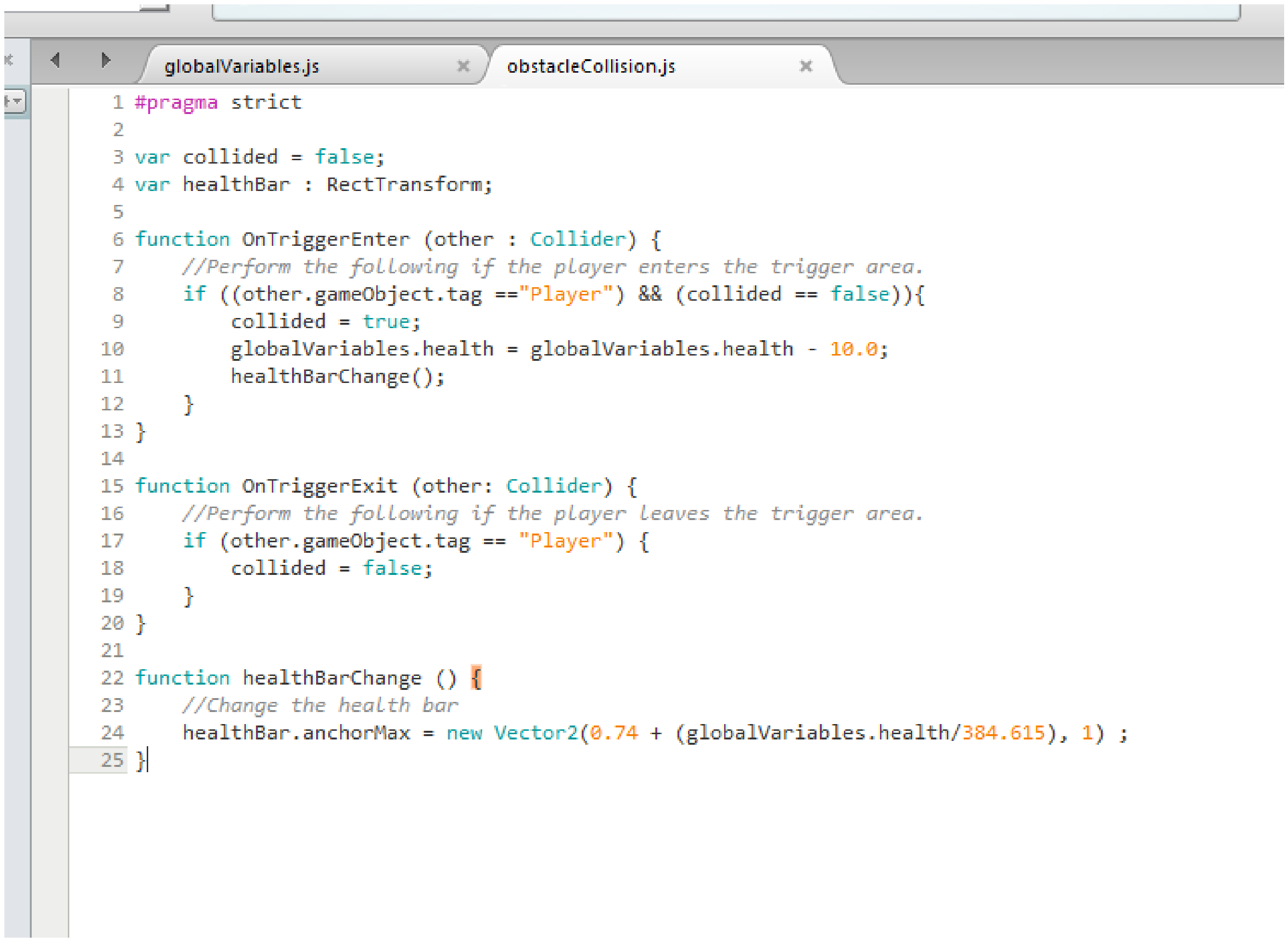
Unity sometimes uses different attributes in the normal inspector in order to simplify editing within Unity. In order to create a proper script we need to know the actual attributes that Unity is using by entering "Debug" mode. Anchor Max is the value we will be modifying in order to resize the health bar.



After playing with the Anchor Max we can find that the smallest it goes before disappearing is approximately 0.74.

Now open MonoDevelop and type the following:

Unity Tutorials 8 – Basic Scripting



```
 1 #pragma strict
 2 
 3 var collided = false;
 4 var healthBar : RectTransform;
 5 
 6 function OnTriggerEnter (other : Collider) {
 7     //Perform the following if the player enters the trigger area.
 8     if ((other.gameObject.tag =="Player") && (collided == false)){
 9         collided = true;
10         globalVariables.health = globalVariables.health - 10.0;
11         healthBarChange();
12     }
13 }
14 
15 function OnTriggerExit (other: Collider) {
16     //Perform the following if the player leaves the trigger area.
17     if (other.gameObject.tag == "Player") {
18         collided = false;
19     }
20 }
21 
22 function healthBarChange () {
23     //Change the health bar
24     healthBar.anchorMax = new Vector2(0.74 + (globalVariables.health/384.615), 1) ;
25 }
```

Let's go over this code line by line.

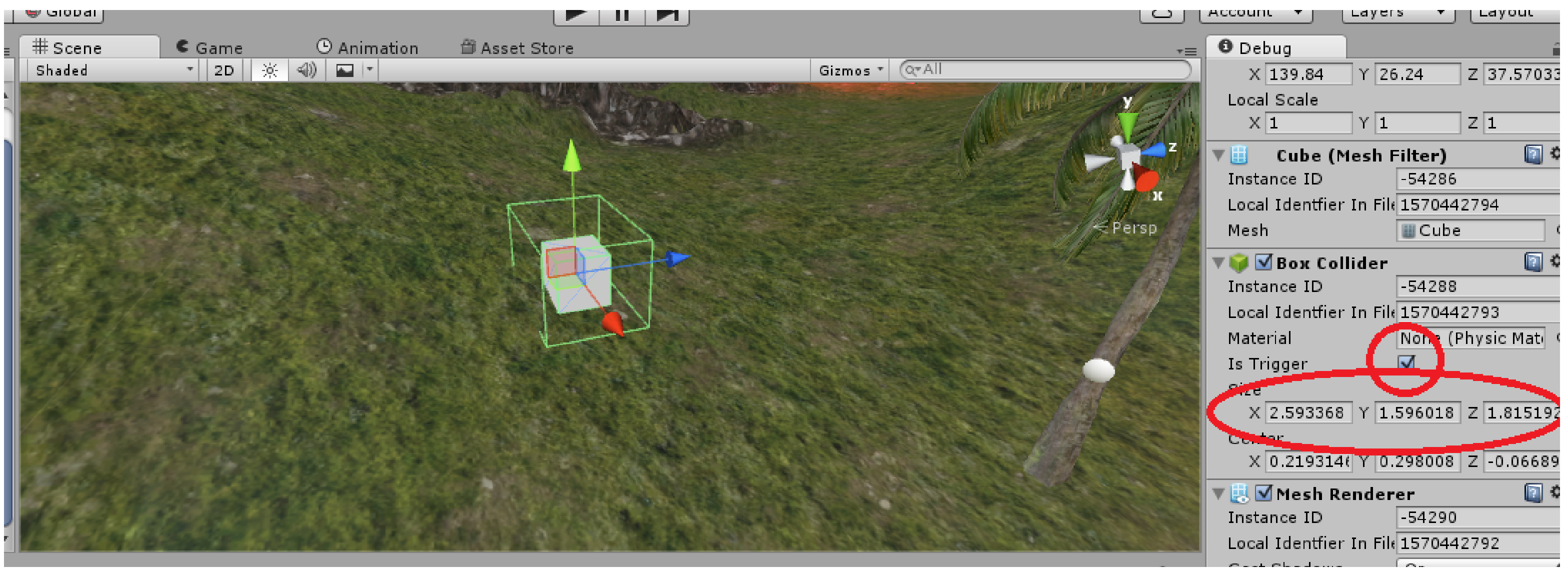
3. Here we are declaring a boolean called “collided”. This will be used to determine whether or not the player is colliding with the obstacle.
4. The healthBar variable refers to the RawImage (1) GameObject. Obviously Unity cannot tell exactly what GameObject “healthBar” is referring to using this script alone. We will assign our RawImage (1) to this variable within Unity.
6. This is the function that will be activated if the FPSController enters a trigger box collider of the object that this script is assigned to. Notice how we use a bracket at the end instead of a semicolon. Anything inside the brackets are part of the function. You can see by the closing bracket on line 13 that the function ends there.
7. Two slashes turn an entire line into a comment. Comments are used to leave notes within a code. /*Anything between a slash and an asterisk will also be a comment. This method is used to make a comment span multiple lines*/.
8. This is the beginning of an “if” statement. Anything within this statement's brackets will only be executed if the described conditions are met. You can see that the conditions

are ((other.gameObject.tag =="Player") && (collided==false)). The && means “and” while a || means or. Because we are using && both conditions need to be met. If || was used instead only one of the conditions would need to be met. “other.gameObject.tag” is the tag that the colliding object has. We are going to change the FPSController's tag to “Player”. This script will also only run if collided is equal to false. This way it will not run over and over as long as the player is within the obstacle's box collider.
9. Collided is set to true so that both conditions for this function to run are no longer met. Notice that only one = is used here. “==” is used for booleans. It means “is already equal to”. “=” is used to assign values to a variable. It means “is now equal to”.
10. We need to use globalVariables.health in order to reach our “health” variable in the “globalVariables” script. In this line “health” is now equal to itself, minus 10.
11. The function “healthBarChange” below is called in order to make the appropriate changes to RawImage (1).
15. This is the start of our “OnTriggerExit” function. In this function “collided” is set back to false so that “OnTriggerEnter” will be fully executed again if the player approaches the obstacle again.
24. healthBar.anchorMax means that we are changing the “Anchor Max” variable of the “healthBar” object variable we defined at the beginning of this code. You do not need to understand “Vector2” at the moment, but take a look at the values we used. The first is (0.74 + (globalVariables.health/384.615). We use 0.74 because that is the lowest value it can have before disappearing. Let's say “health” is at 100. We use 384.615 because 100/384.615 is equal to 1, the max value that we want the Anchor Max to have. (1-0.74 = 0.26; 100/0.26 = 384.615). The second value, 1, is the Y Anchor Max. We only want to change the X Anchor Max so we leave it at 1.

Finally, notice that we use decimals throughout the code. This is to prevent Unity from assigning integer values (whole numbers) to the Anchor Max as opposed to floats (fractional numbers).

Now save the code and return to Unity. Create a cube and give it a new Box Collider. Turn on “Is Trigger” and make the box slightly larger using the Size fields if you are still in “Debug” mode. Otherwise click the “Edit Trigger” button.

Unity Tutorials 8 – Basic Scripting



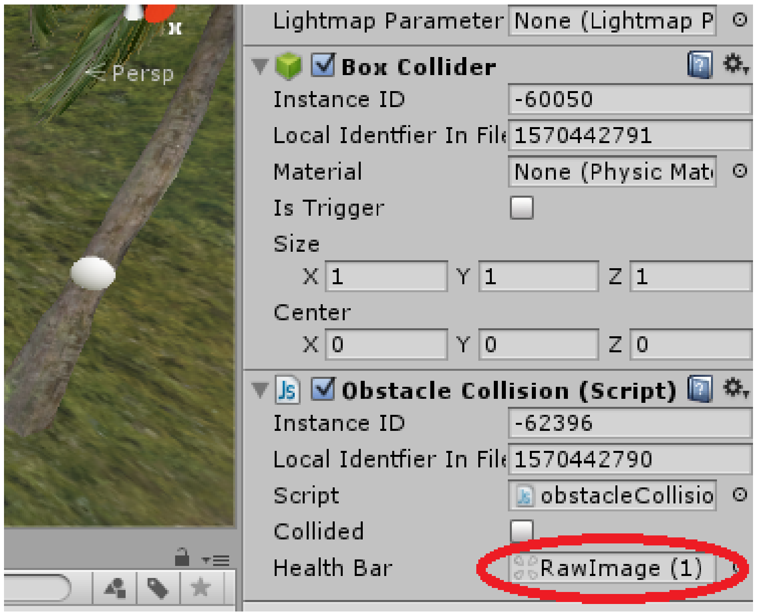
Local Scale X 1 Y 1 Z 1

Cube (Mesh Filter)
Instance ID -54286
Local Identfier In File 1570442794
Mesh Cube

Box Collider
Instance ID -54288
Local Identfier In File 1570442793
Material None (Physic Mat
Is Trigger ✓
Size
X 2.593368 Y 1.596018 Z 1.815192
Center
X 0.219314 Y 0.298008 Z -0.06689

Mesh Renderer
Instance ID -54290
Local Identfier In File 1570442792

Now click and drag your obstacleCollision script onto the object either in the hierarchy or scene view. It will appear in the inspector.



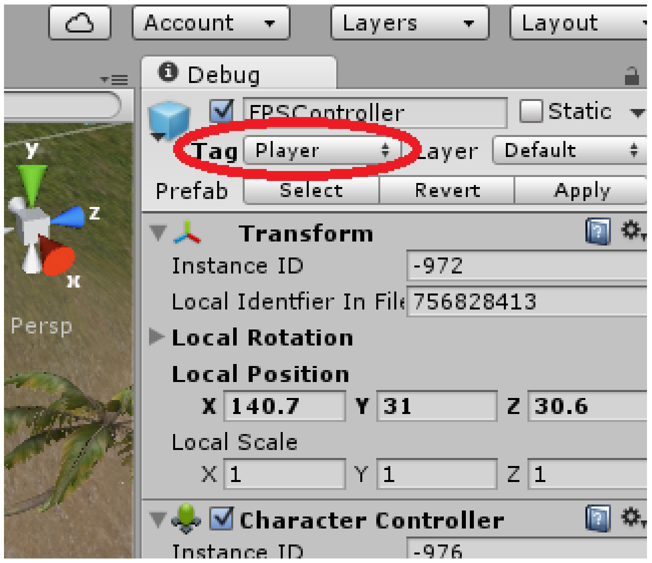
Lightmap Parameter None (Lightmap P

Box Collider
Instance ID -60050
Local Identfier In File 1570442791
Material None (Physic Mat
Is Trigger ☐
Size
X 1 Y 1 Z 1
Center
X 0 Y 0 Z 0

Obstacle Collision (Script)
Instance ID -62396
Local Identfier In File 1570442790
Script obstacleCollisio
Collided ☐
Health Bar RawImage (1)

Set the “Health Bar” field to “RawImage (1)”.

Open your FPSController and change the “Tag” to “Player”.



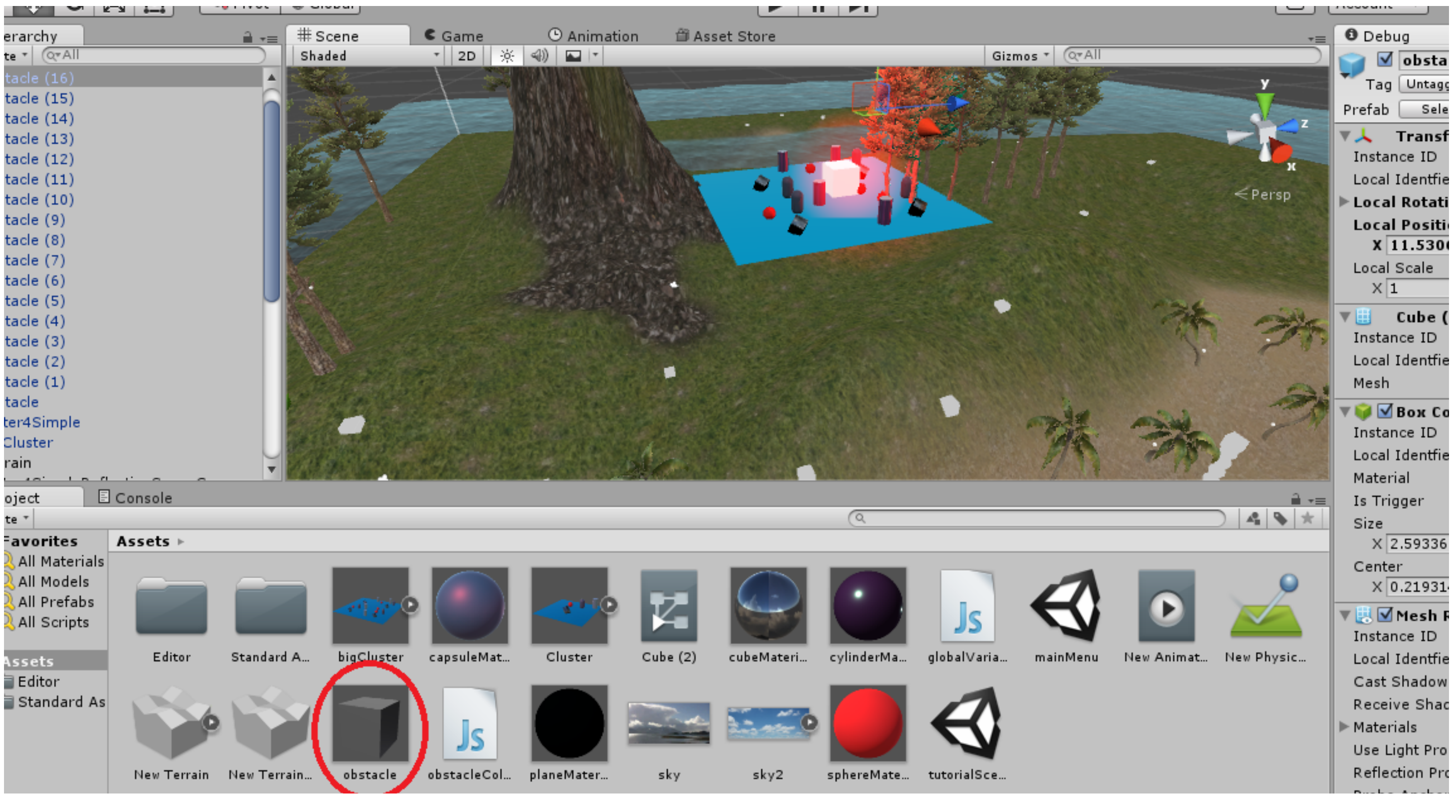
FPSController ☐ Static
Tag Player Layer Default
Prefab Select Revert Apply

Transform
Instance ID -972
Local Identfier In File 756828413
Local Rotation
Local Position
X 140.7 Y 31 Z 30.6
Local Scale
X 1 Y 1 Z 1

Character Controller
Instance ID -976

Now create a prefab out of your obstacle and place them throughout the scene.

Unity Tutorials 8 – Basic Scripting



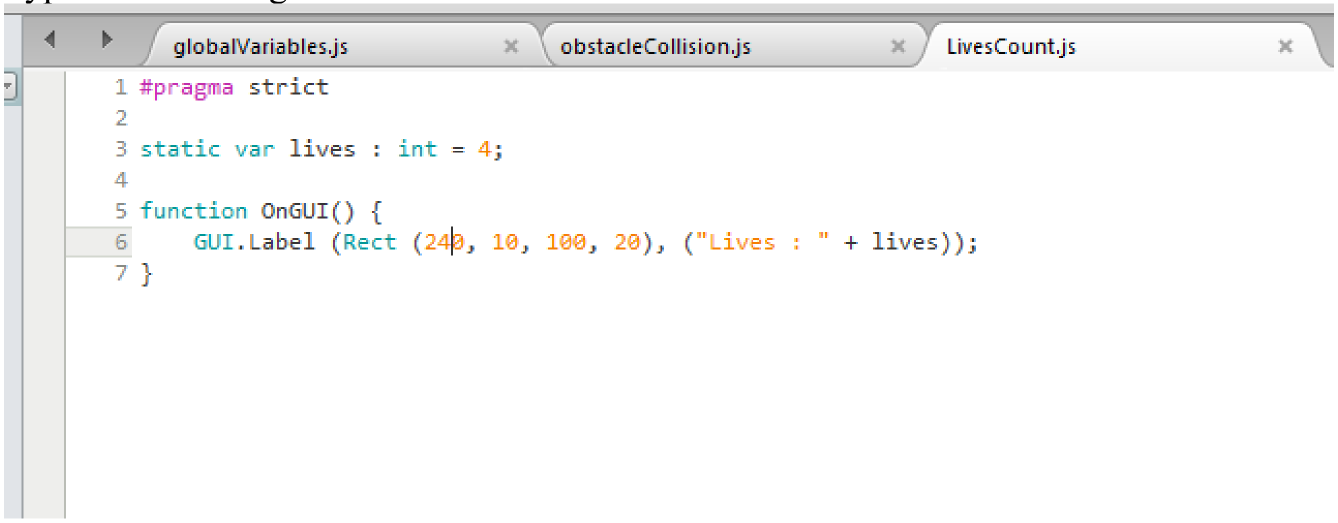
You may need to individually assign the “RawImage (1)” object to each obstacle.

Unity Tutorials 8 – Basic Scripting

III – Lives Count

Create a new Javascript called “LivesCount” and open it in MonoDevelop.

Type the following:

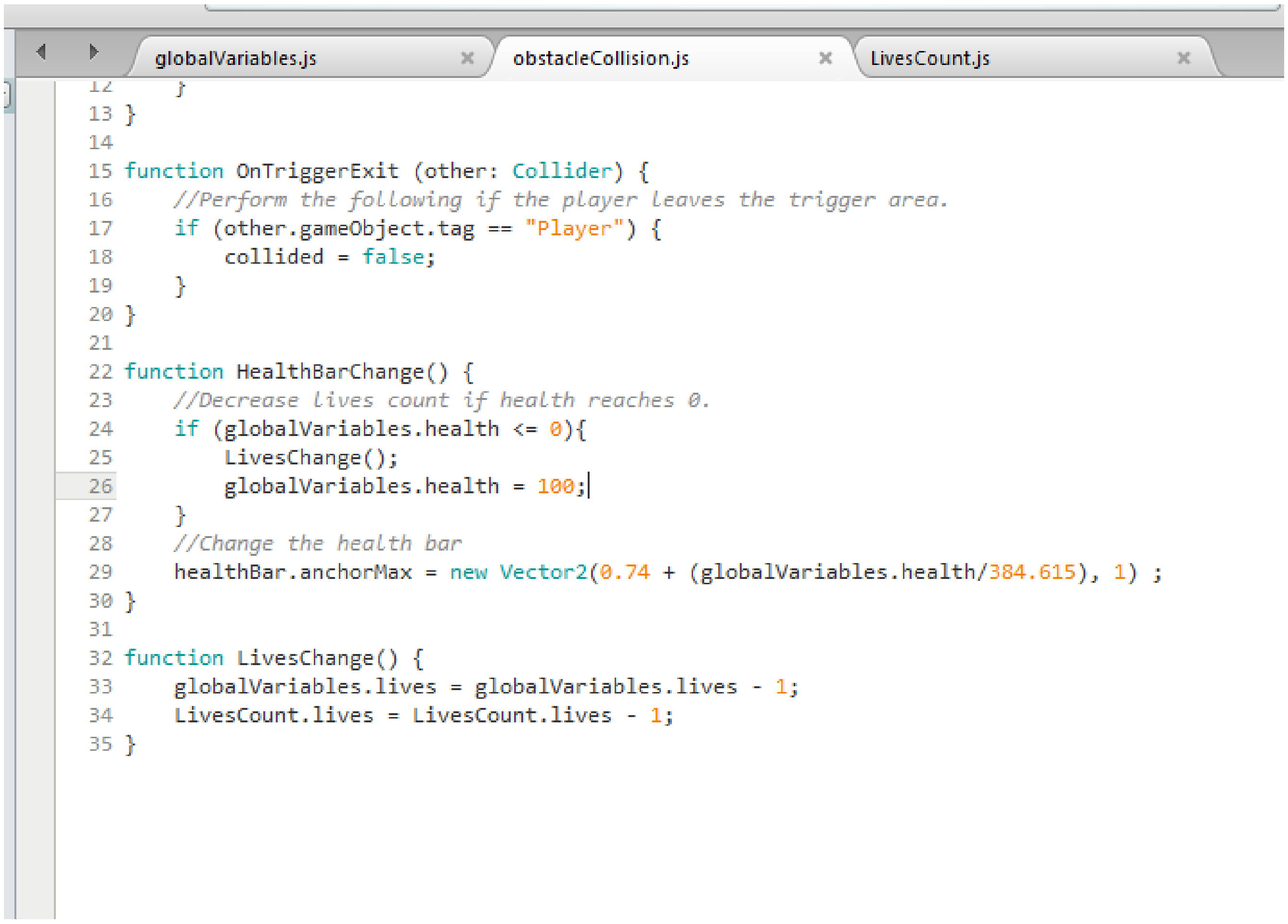


```
#pragma strict

static var lives : int = 4;

function OnGUI() {
    GUI.Label (Rect (240, 10, 100, 20), ("Lives : " + lives));
}
```

Here we are creating a GUI label that is automatically updated whenever the “lives” variable updates.

You will need to add this script to the livesCounter object by clicking and dragging it onto it.

Now make the following changes to “obstacleCollision.js”:



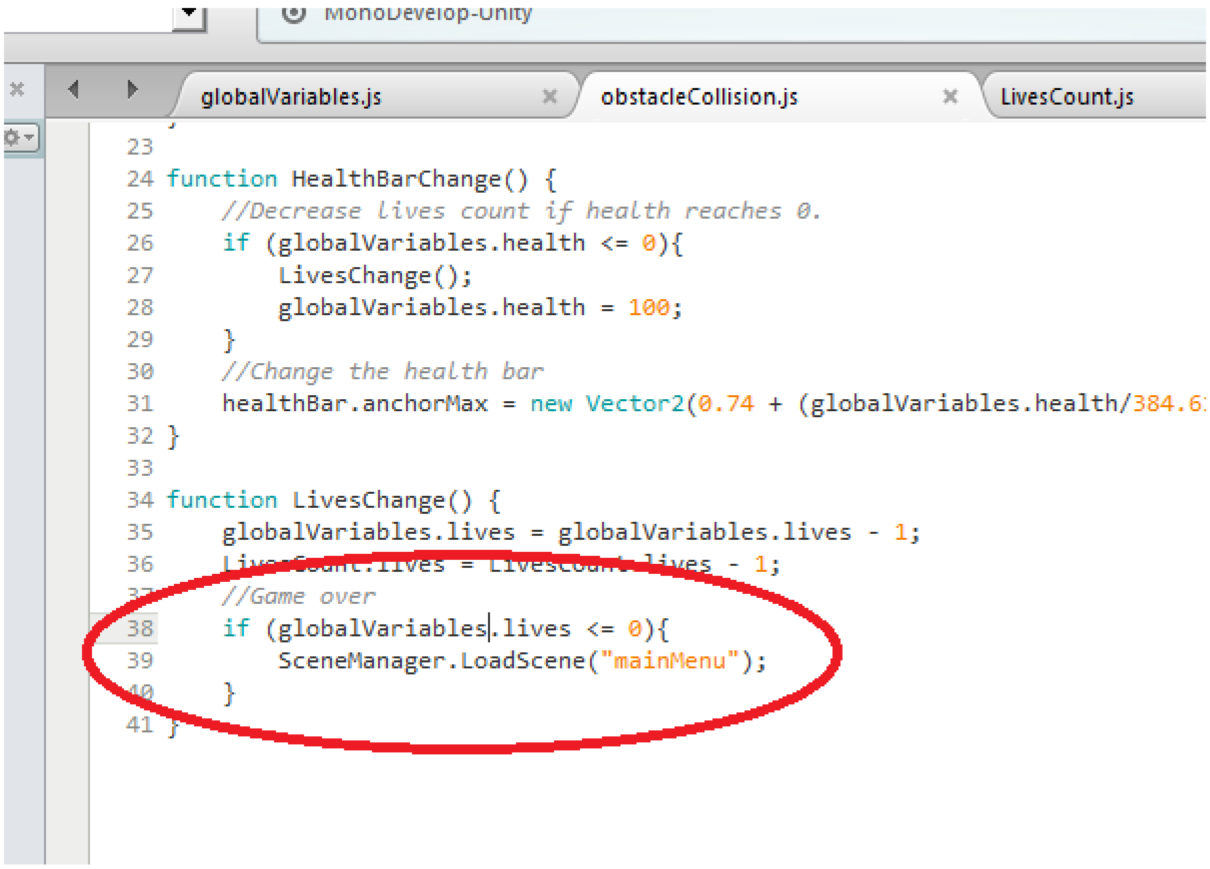
```
    }
}

function OnTriggerExit (other: Collider) {
    //Perform the following if the player leaves the trigger area.
    if (other.gameObject.tag == "Player") {
        collided = false;
    }
}

function HealthBarChange() {
    //Decrease lives count if health reaches 0.
    if (globalVariables.health <= 0){
        LivesChange();
        globalVariables.health = 100;
    }
    //Change the health bar
    healthBar.anchorMax = new Vector2(0.74 + (globalVariables.health/384.615), 1) ;
}

function LivesChange() {
    globalVariables.lives = globalVariables.lives - 1;
    LivesCount.lives = LivesCount.lives - 1;
}
```

Here we added to the HealthBarChange function to make it detect when the player's health reaches 0. It then calls a new function, “LivesChange”, which decreases the player's lives count both in the globalVariables class and in the LivesCount class. As stated before, when the LivesCount's “lives” variable changes the GUI will automatically update. HealthBarChange will also replenish the player's health.

Try playing your scene and running into obstacles until your health runs out. You can also change how much health the player loses to 20.0 to speed up the process.

We now need to add a “game over” function. Add the following to “obstacleCollision”:

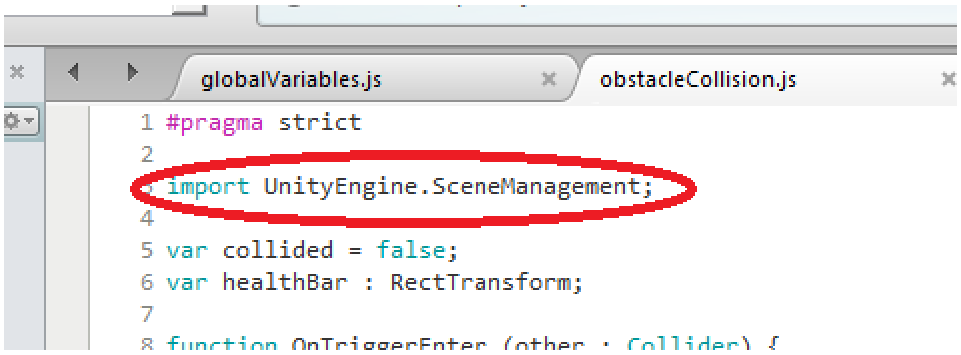
Unity Tutorials 8 – Basic Scripting



MonoDevelop-Unity

```
function HealthBarChange() {
    //Decrease lives count if health reaches 0.
    if (globalVariables.health <= 0){
        LivesChange();
        globalVariables.health = 100;
    }
    //Change the health bar
    healthBar.anchorMax = new Vector2(0.74 + (globalVariables.health/384.6
}

function LivesChange() {
    globalVariables.lives = globalVariables.lives - 1;
    LivesCount.lives = LivesCount.lives - 1;
    //Game over
    if (globalVariables.lives <= 0){
        SceneManager.LoadScene("mainMenu");
    }
}
```

You will also need to add to the beginning.



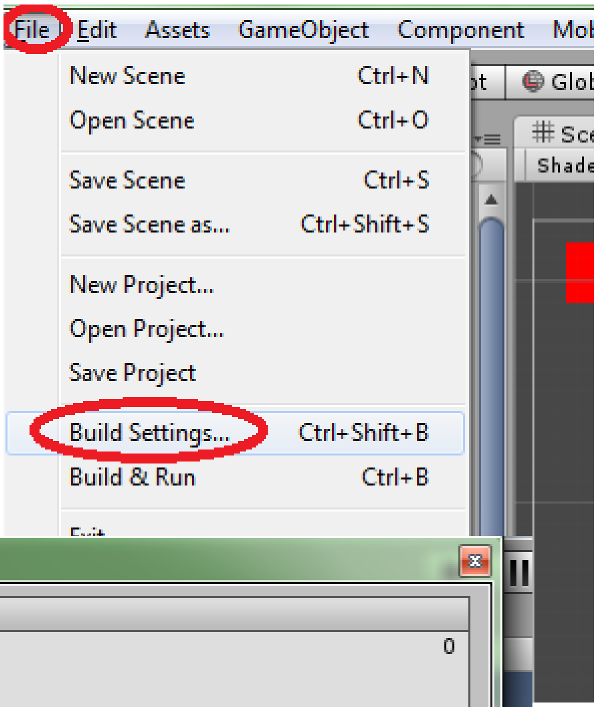
```
#pragma strict

import UnityEngine.SceneManagement;

var collided = false;
var healthBar : RectTransform;

function OnTriggerEnter (other : Collider) {
```

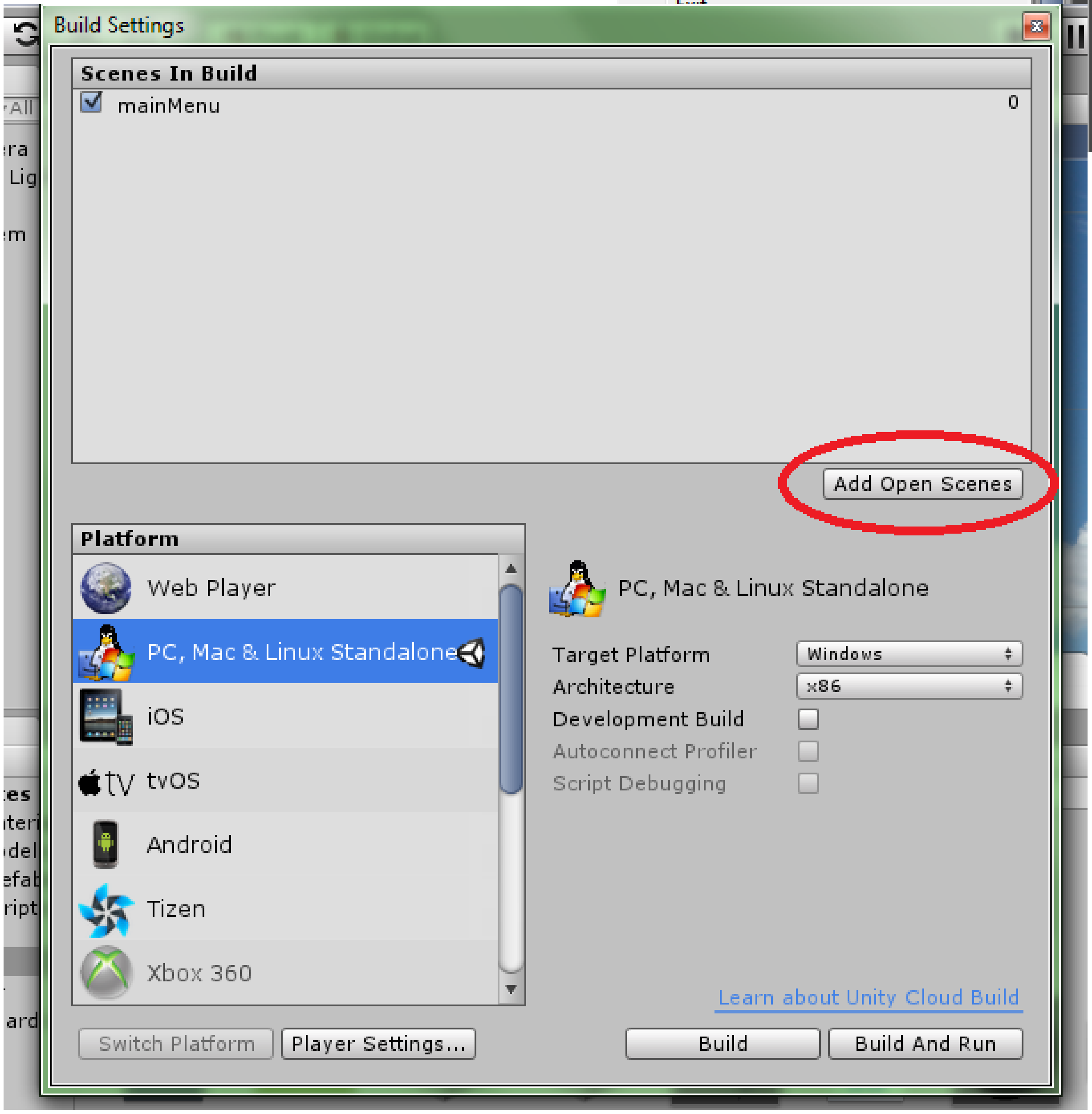
Now before you can load another scene you will need to make some adjustments that will be covered in greater detail in the next tutorial.

Open your mainMenu scene and click “File” and “Build Settings”.

Unity Tutorials 8 – Basic Scripting

Now click "Add Open Scenes". Open your level and do the same thing. You can then test your code.





Scenes In Build

✓ mainMenu	0

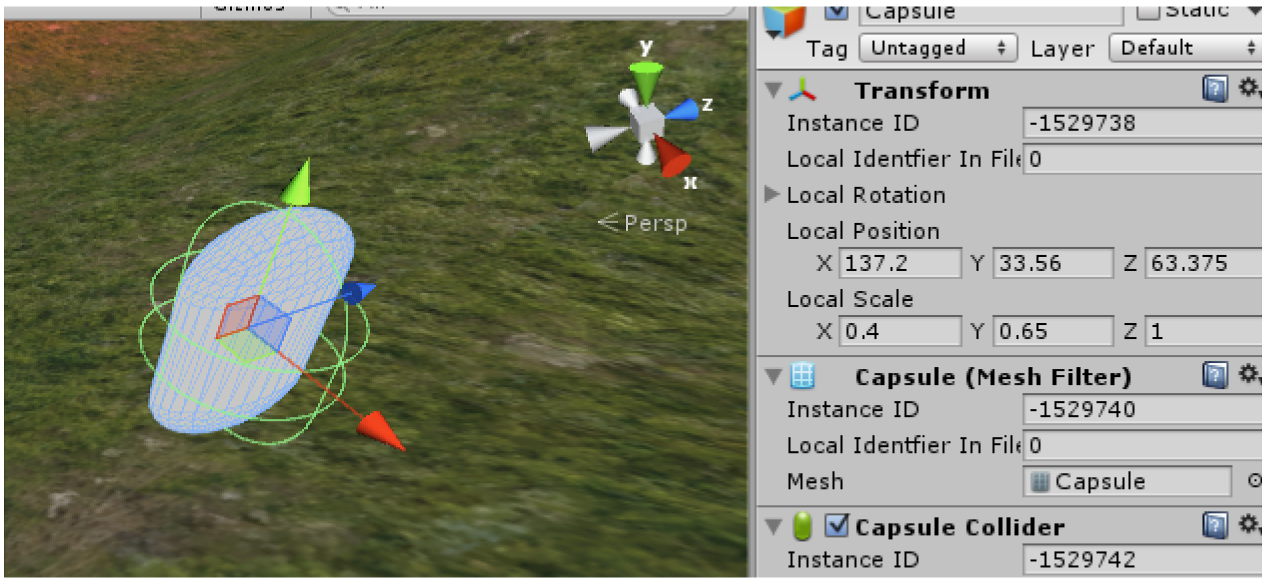
PC, Mac & Linux Standalone

Target Platform: Windows
Architecture: x86
Development Build: ☐
Autoconnect Profiler: ☐
Script Debugging: ☐

Unity Tutorials 8 – Basic Scripting

IV – Coin Count

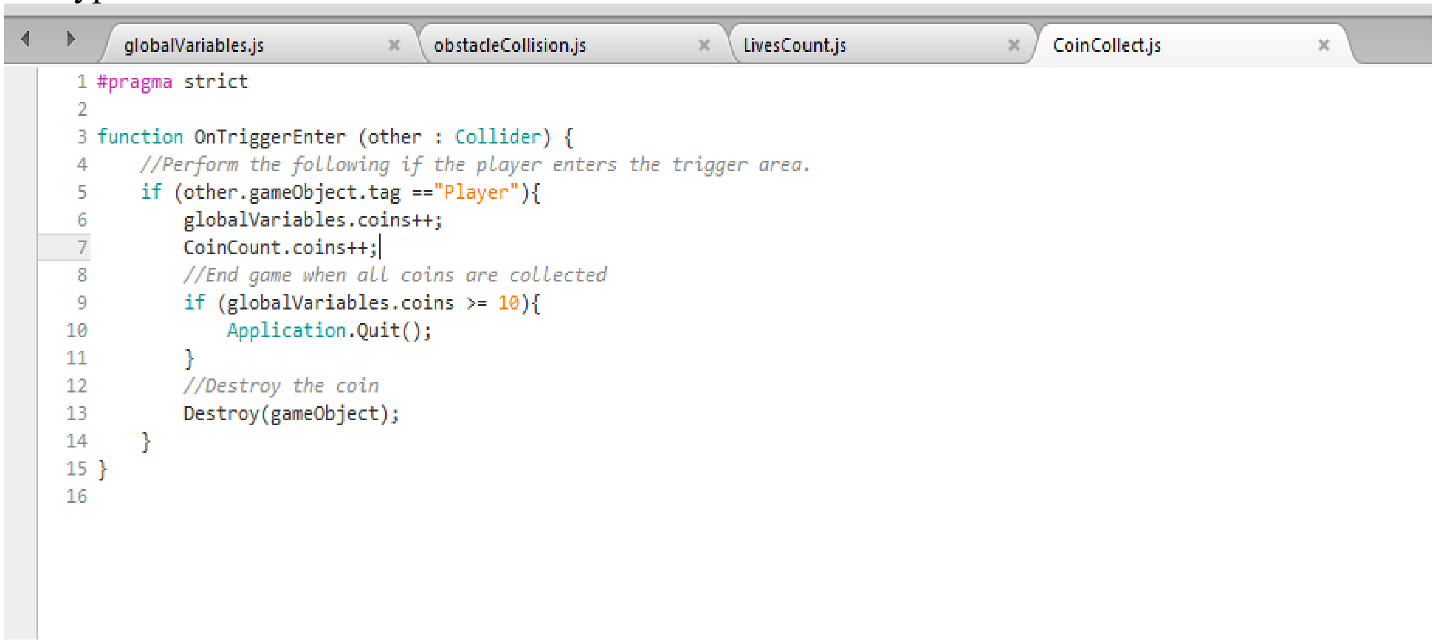
We are now going to make a code that is very similar to the last one. First make a sphere or capsule object and shape it into a coin.



Capsule Static
Tag Untagged Layer Default
Transform
Instance ID -1529738
Local Identfier In File 0
Local Rotation
Local Position
X 137.2 Y 33.56 Z 63.375
Local Scale
X 0.4 Y 0.65 Z 1
Capsule (Mesh Filter)
Instance ID -1529740
Local Identfier In File 0
Mesh Capsule
Capsule Collider
Instance ID -1529742

Create a Javascript file and call it "CoinCollect". Apply it to the capsule and open it up in MonoDevelop.

Type this and save it:



```
#pragma strict

function OnTriggerEnter (other : Collider) {
    //Perform the following if the player enters the trigger area.
    if (other.gameObject.tag =="Player"){
        globalVariables.coins++;
        CoinCount.coins++;
        //End game when all coins are collected
        if (globalVariables.coins >= 10){
            Application.Quit();
        }
        //Destroy the coin
        Destroy(gameObject);
    }
}

```

Unity Tutorials 8 – Basic Scripting

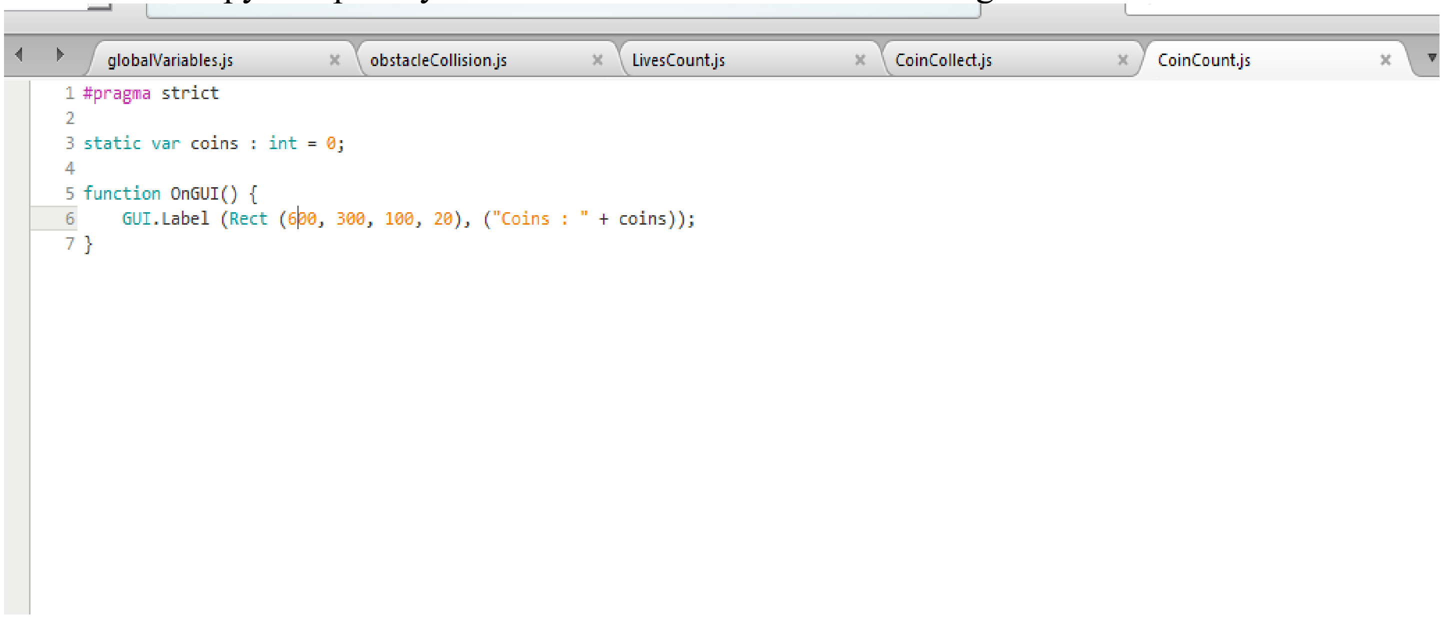
Give your coin a Box Collider and check “Is Trigger”.

Make a prefab called “coin” and spread 10 of them around your map.

Now it's finally time to add the coin counter.

As before, create an empty GameObject and create a new Javascript. Call it “CoinCount” and add it to the GameObject. Call this object “coinCounter”.

You can copy and paste your code from LivesCount and change it as such:



```
#pragma strict

static var coins : int = 0;

function OnGUI() {
    GUI.Label (Rect (600, 300, 100, 20), ("Coins : " + coins));
}
```

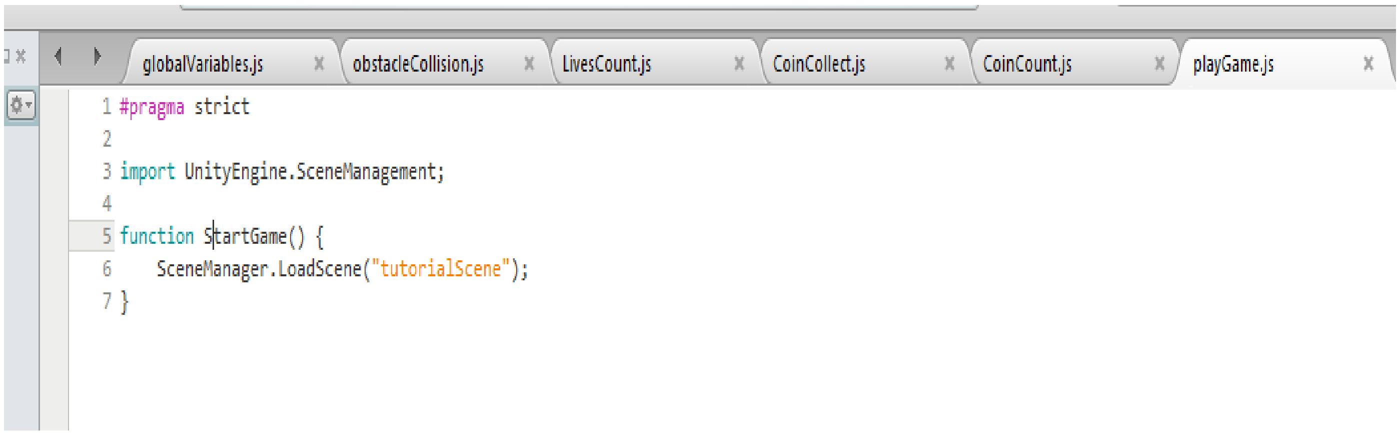
NOTE: The Quit() feature only works during a build and will not actually work while testing in the Game View.

Unity Tutorials 8 – Basic Scripting

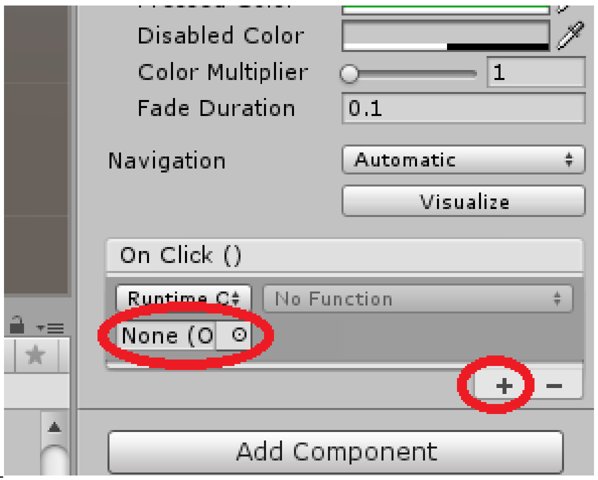
V – Buttons

The final step is to add functionality to the “Play” button in the main menu. Start by opening your mainMenu scene.

Create a Javascript called “playGame”.



```
#pragma strict

import UnityEngine.SceneManagement;

function StartGame() {
    SceneManager.LoadScene("tutorialScene");
}
```

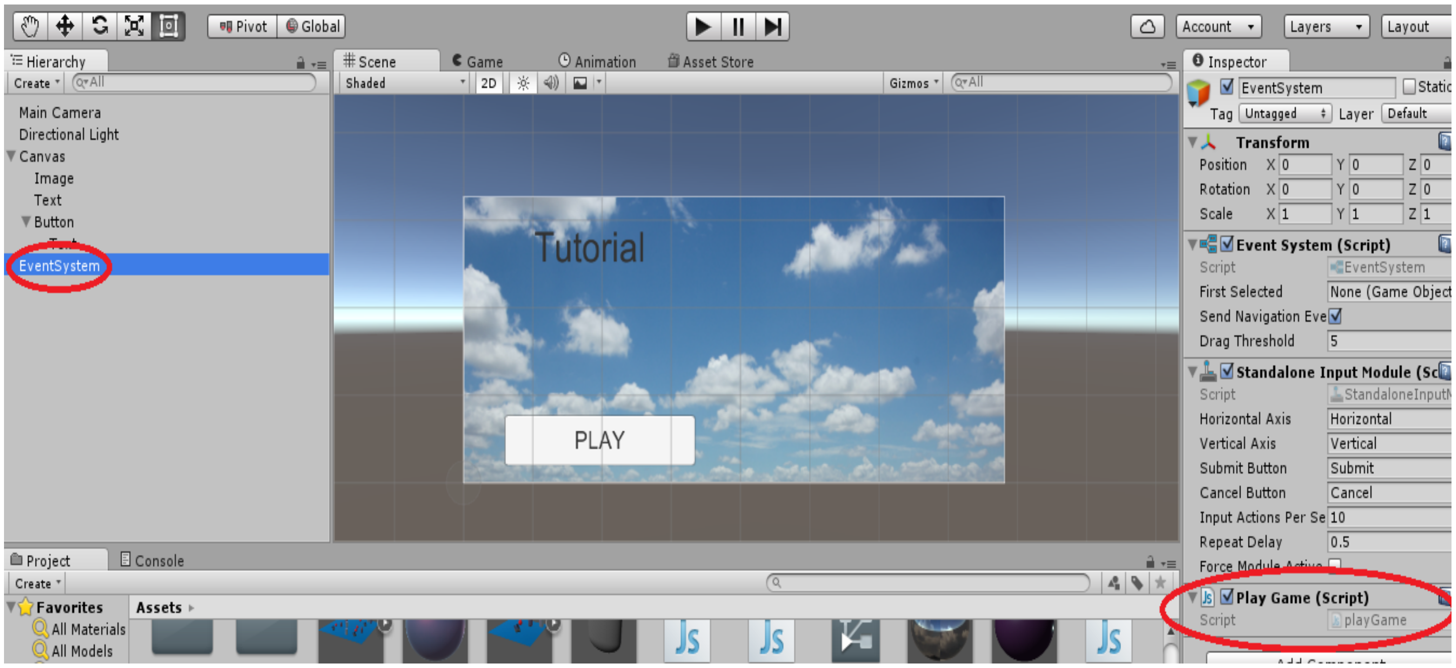


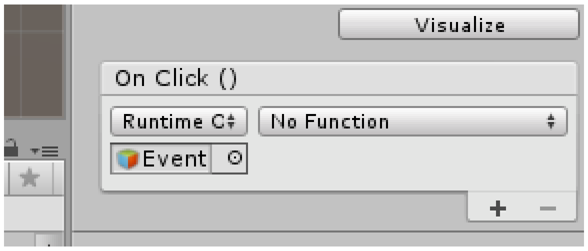
If you named your scene something else you will need to adjust the script accordingly.

Now click on your button and make sure the inspector is in “Normal” mode and not “Debug” mode. Click the “+” in the On Click () menu.

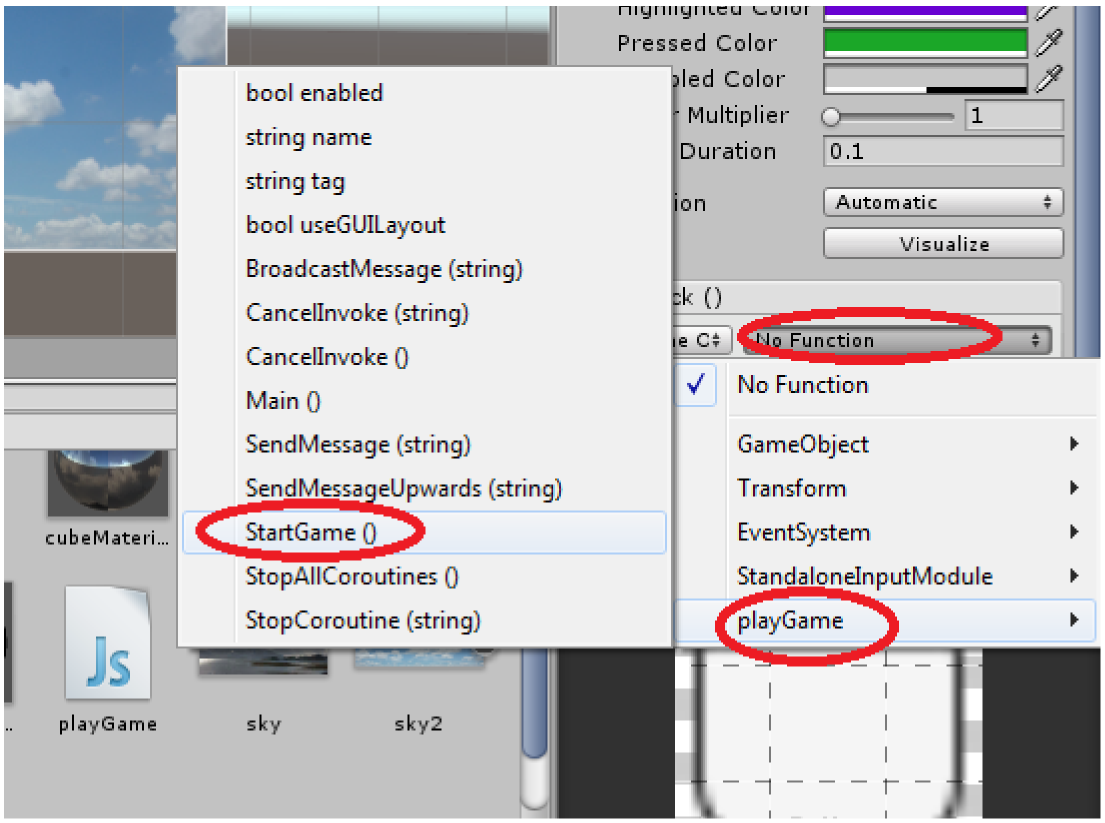
It's hard to see, but there is a field here that says None (Object). You will need to first apply the script to another object and apply that object to this field. Thankfully, there should be an object in your scene called “EventSystem”. Drag your playGame script onto this object then apply this object to the field under On Click ().

Unity Tutorials 8 – Basic Scripting





Now click the “No Function” box and click on “playGame” and “StartGame ()”.


bool enabled
string name
string tag
bool useGUILayout
BroadcastMessage (string)
CancelInvoke (string)
CancelInvoke ()
Main ()
SendMessage (string)
SendMessageUpwards (string)
StartGame ()
StopAllCoroutines ()
StopCoroutine (string)
No Function
No Function
GameObject
Transform
EventSystem
StandaloneInputModule
playGame
Pressed Color
Automatic
Visualize
playGame
sky
sky2