

Scripting

MONO compiler / Visual Studio (2018+)

Scripts can be written in

C#

Unity can be integrated with the Microsoft Visual Studio editor, to get full benefits of code completion, source version control, integration, serious developers work in C#

JavaScript

Majority of introductory tutorials are written in Javascript

BOO (like Python)

Smaller development in this

Scripting

scripting is Unity's most powerful tool

gives you the ability to customize objects

control how they behave in the environment

- how to create and attach JavaScript scripts to objects in Unity
- Intro to the development environment MonoDevelop/Visual Studio

Scripting

Variables

Functions

Syntax

Arithmetic operators

If statement

Sounds

Colors

JavaScript vs C#

JavaScript

```
#pragma strict
```

```
var myInt : int = 5;
```

```
function Start ()
```

```
{
```

```
    myInt = MultiplyByTwo(myInt);
```

```
    Debug.Log (myInt);
```

```
}
```

C#

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

```
public class VariablesAndFunctions  
    : MonoBehaviour
```

```
{
```

```
    int myInt = 5;
```

```
    void Start ()
```

```
{
```

```
        myInt = MultiplyByTwo(myInt);
```

```
        Debug.Log (myInt);
```

```
}
```

Scripting

You can use both C# and Javascript in one project!
(one way communication only)

My Scripts Folder (Outside)
(Compiled last)

Script
Script
script

JavaScript

Standard Assets
(Compiled first))

Script
Script
Script

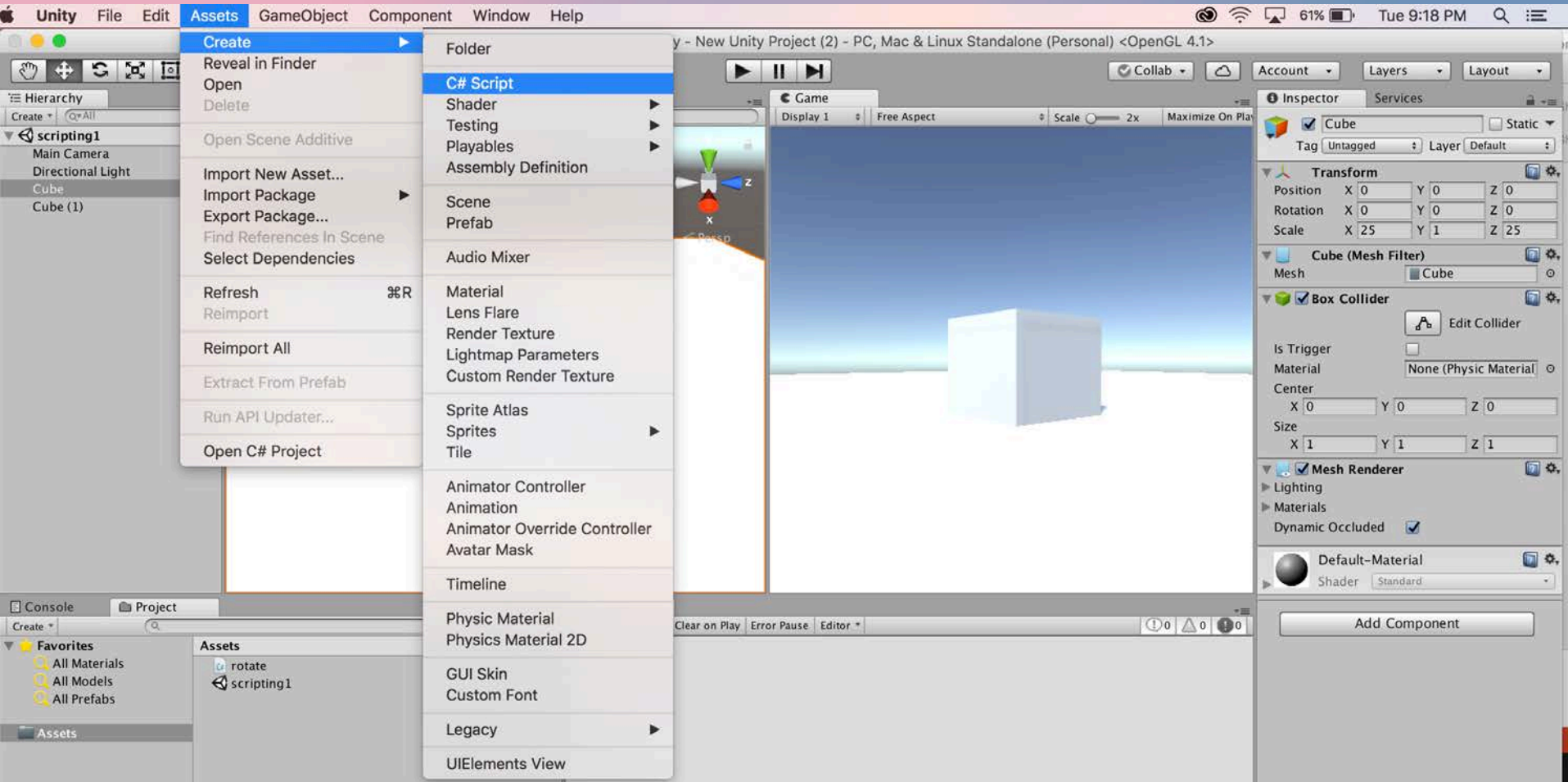
C#

Creating scripts in Unity

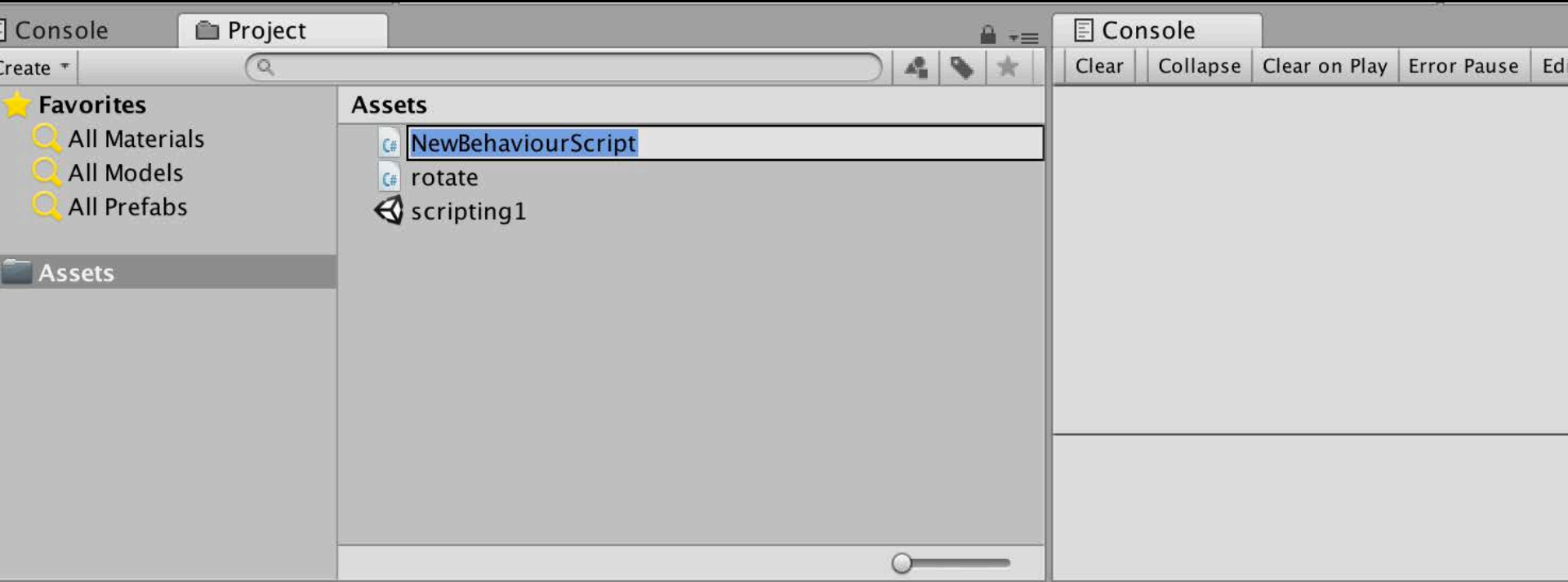
- Project menu >Create > C# Script
- Main Menu > Assets > Create C# Script
- Project window >RMC > Create > C# Script
- Inspector >Add script
- Name the script in the Project/Assets window

- Assign the script to an object (drag and drop)
- Run and test
- Fix compiler errors

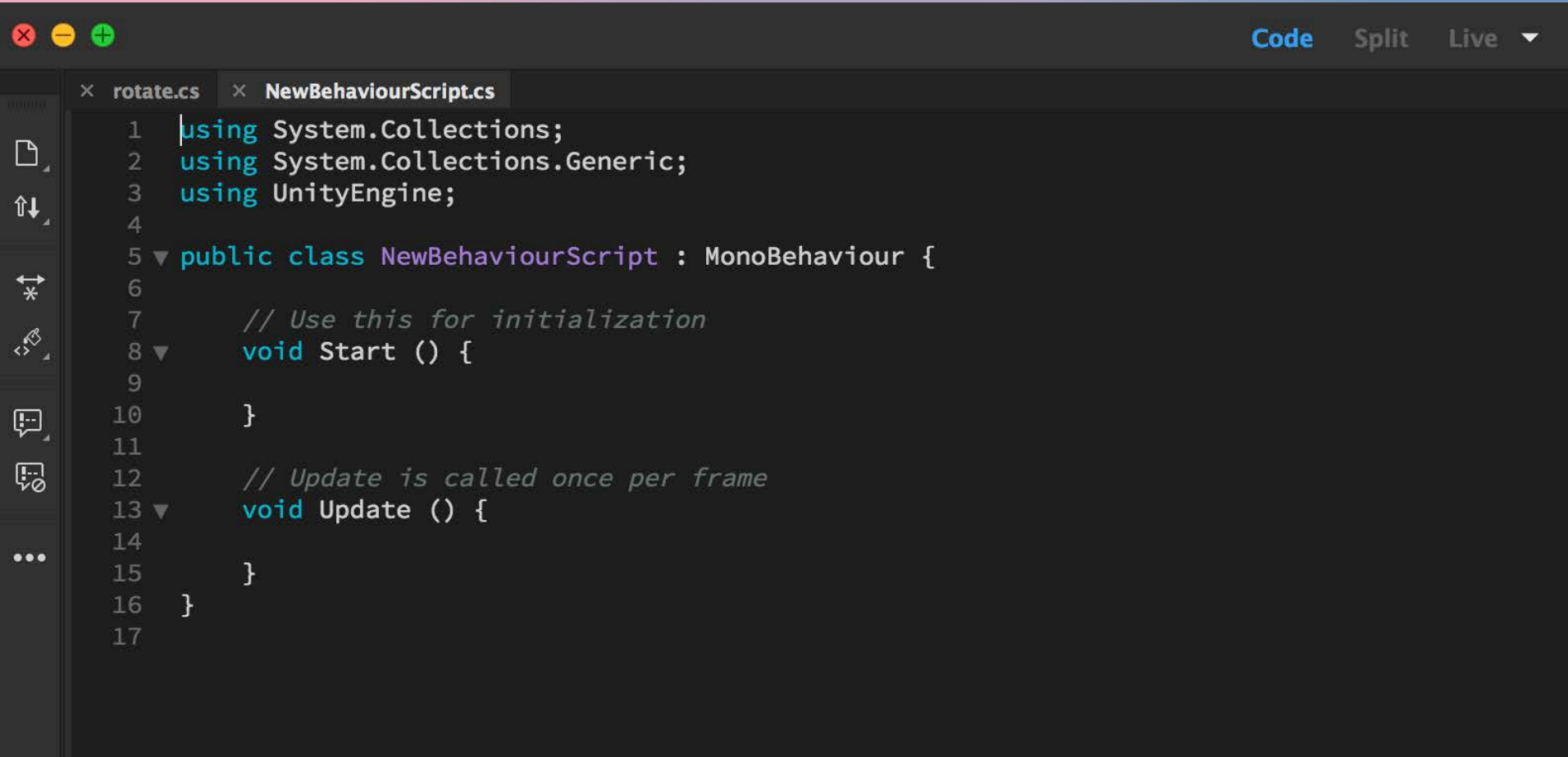
Creating scripts in Unity



Creating scripts in Unity



Creating scripts in Unity



The image shows a code editor window with two tabs: 'rotate.cs' and 'NewBehaviourScript.cs'. The 'NewBehaviourScript.cs' tab is active, displaying the following C# code:

```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class NewBehaviourScript : MonoBehaviour {
6
7     // Use this for initialization
8     void Start () {
9
10    }
11
12    // Update is called once per frame
13    void Update () {
14
15    }
16 }
17
```

The code defines a class named 'NewBehaviourScript' that inherits from 'MonoBehaviour'. It includes two methods: 'Start' and 'Update'. The 'Start' method is currently empty, and the 'Update' method is also empty. The code is color-coded: 'using' statements are in blue, 'public class' is in purple, 'MonoBehaviour' is in light blue, and 'void' is in cyan. Comments are in a lighter grey font.

x variables.cs

```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class variables : MonoBehaviour {
6
7     int myInt = 5;
8
9
10    void Start ()
11    {
12        myInt = MultiplyByTwo(myInt);
13        Debug.Log (myInt);
14    }
15
16
17    int MultiplyByTwo (int number)
18    {
19        int ret;
20        ret = number * 2;
21        return ret;
22    }
23 }
```

Center Local

Hierarchy

- Create - All
- Main Camera
- GameObject

Scene

Textured RGB 2D Effects Gizmos All

Game

Free Aspect Maximize on Play

Inspector

Layers Layout

Y X Z

← Persp

All compiler errors have to be fixed before you can enter playmode!

Console

Project

Assets

- Variables

Console

Clear Collapse Clear on Play Error Pause

10

UnityEngine.Debug.Log(Object)

Assets/Variables.js(18,13): BCE0005: Unknown identifier: 'MultiplyByThree'

Center Local

Hierarchy: Main Camera, GameObject

Scene: Textured, RGB, 2D, Effects, Gizmos, OcAll

Game: Free Aspect, Maximize on Play, St

Inspector

Layers Layout

A 3D scene view showing a perspective view of a gray grid floor. A white camera icon is positioned in the center. A 3D coordinate system with x, y, and z axes is visible in the top right corner. A semi-transparent message box in the center of the grid reads: "All compiler errors have to be fixed before you can enter playmode!".

Console

Project: Variables

Console: Clear, Collapse, Clear on Play, Error Pause

- 10 UnityEngine.Debug.Log(Object)
- Assets/Variables.js(8,13): BCE0005: Unknown identifier: 'MultiplyByThree'.

Assets

The console window displays two messages. The first is a log message: "10 UnityEngine.Debug.Log(Object)". The second is an error message: "Assets/Variables.js(8,13): BCE0005: Unknown identifier: 'MultiplyByThree'." The error message is highlighted with a red icon. The project window shows a folder named "Variables" under the "Assets" category.

Variables

- A variable is a storage location and an associated symbolic name (an identifier) which contains some known or unknown quantity or information, a value
- variables are used to store information about any aspects of a project's state

Variables

begin with a lowercase letter

no special characters, numbers, (#, %, etc.)

cannot contain reserved keywords such as “if”, “while”, etc.

case sensitive

descriptive

no spaces

Type/ Declaration/ Initialization

```
int myInt = 5;
```

Selected Data Types

Float	0.75
Int	10
String	“Hello”
Boolean	true / false

```
int myInt = 5;  
float speed = 10.5;
```

Variables

× variables.cs

```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class variables : MonoBehaviour {
6
7     int myInt = 5;
8
9
10    void Start ()
11    {
12
13        Debug.Log (myInt);
14    }
15
16
17 }
```


Variables

```
public class variables : MonoBehaviour {  
    int myInt = 5;  
  
    void Start ()  
    {  
        int myInt = 55;  
        Debug.Log (myInt * 2);  
    }  
}
```

Functions

Function is a collection of statements to perform a task

Known as Method

Functions are blocks of code which are written once and can then be reused as often as needed.

begin with an uppercase letter

```
type FuncName ( )  
    {  
        statement1;  
        statement 2;  
    }
```

Functions

Calling a function:

```
FuncName ( );
```

```
myInt = MultiplyByTwo(myInt);
```

Function Parameters

```
int MultiplyByTwo (int number)
{
    int ret;
    ret = number * 2;
    return ret;
}
```

Calling a function – `myInt = MultiplyByTwo(myInt);`

Functions

Default functions

Start ()

executed only once before gameplay begins
helpful for initialization

Update()

executed every frame
for as long as the gameplay continues

Functions

```
public class variables : MonoBehaviour {  
    int myInt = 5;  
    void Start ()  
    {  
        myInt = MultiplyByTwo(myInt);  
        Debug.Log (myInt);  
    }  
    int MultiplyByTwo (int number)  
    {  
        int ret;  
        ret = number * 2;  
        return ret;  
    }  
}
```

Arithmetic Operators

+	addition	
-	subtraction	
/	division	
*	multiplication	
++	Increment	<code>x++; x=x+1;</code>
--	Decrement	<code>y--; y=y-1;</code>
%	modulus	<code>5%3=2; 7%4=3; 18%9=0; 17%5=2;</code>

Functions

- 1) Create 3D object cube
- 2) create new C# "rotateCube"
- 3) Assign the script to the cube (drag and drop)

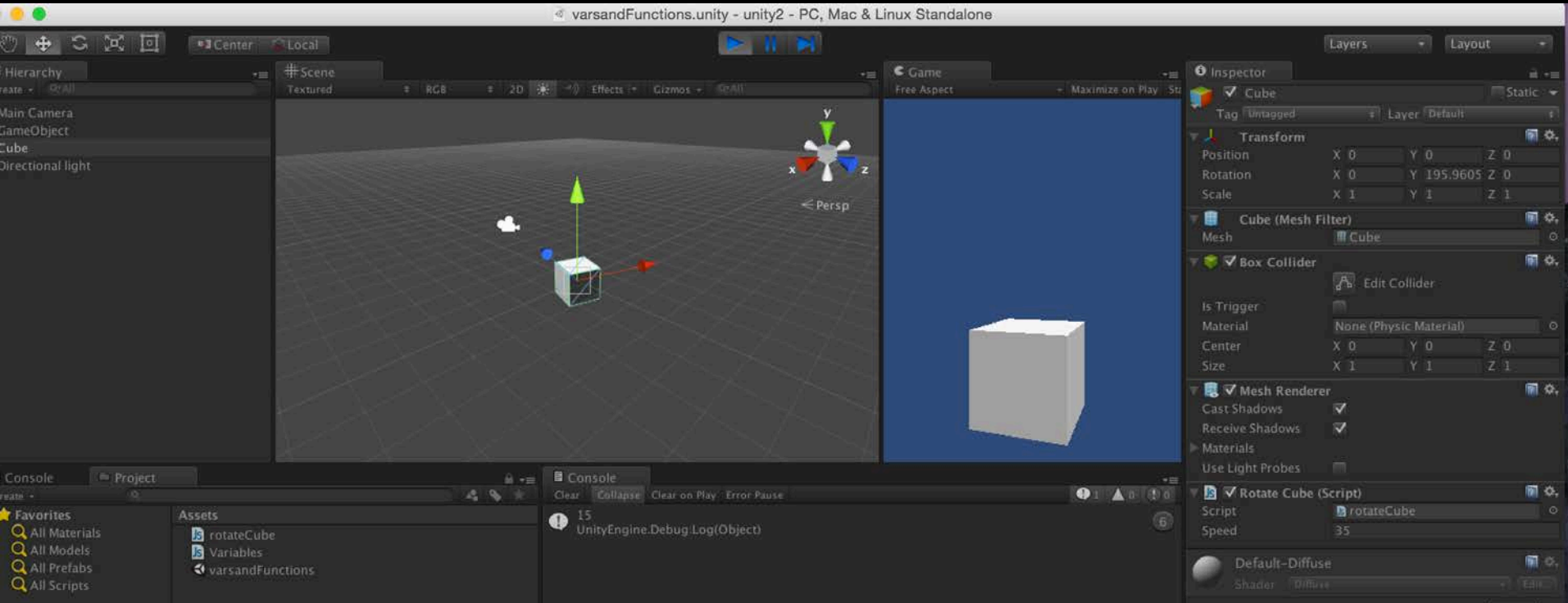
```
public class rotate : MonoBehaviour {  
    void Start ()  
    {  
    }  
    void Update ()  
    {  
transform.Rotate(Vector3.up, 10 * Time.deltaTime);  
    }  
}
```


Functions

```
public class rotate : MonoBehaviour {  
    public float speed = 10;  
    void Start ()  
    {  
    }  
    void Update ()  
    {  
        transform.Rotate(Vector3.up, speed * Time.deltaTime);  
    }  
}
```

Functions

- 4) Change the value of var speed in the Inspector window (35)
- 5) Play and test



Syntax

.operator

; semicolon – end of statement

{ } curly braces

indentation

comments

// single line comment

/* multiple line comment

**

**/

Functions

```
public class rotate : MonoBehaviour {  
    public float speed = 10;  
    void Start ()  
    {  
    }  
    void Update ()  
    {  
        transform.Rotate(Vector3.up, speed * Time.deltaTime);  
    }  
}
```

If statement

Conditional statements are used to perform different actions based on different conditions.

Very often when you write code, you want to perform different actions for different decisions.

You can use conditional statements in your code to do this.

```
if (condition) {  
    block of code to be executed if the condition is true  
}
```

If statement

```
public class ifstatement : MonoBehaviour {  
    void Update() {  
        if (Input.GetKeyDown(KeyCode.C))  
        {  
            GetComponent<Renderer> ().material.color = Color.cyan;  
        }  
        if (Input.GetKeyDown(KeyCode.M))  
        {  
            GetComponent<Renderer>().material.color = Color.magenta;  
        }  
        if (Input.GetKeyDown(KeyCode.Y))  
        {  
            GetComponent<Renderer>().material.color = Color.yellow;  
        }  
    }  
}
```

Sounds

Supported Audio Formats

MPEG layer 3 .mp3

Ogg Vorbis .ogg

Microsoft Wave .wav

Audio Interchange File Format .aiff / .aif

Ultimate Soundtracker module .mod

Impulse Tracker module .it

Scream Tracker module .s3m

FastTracker 2 module .xm

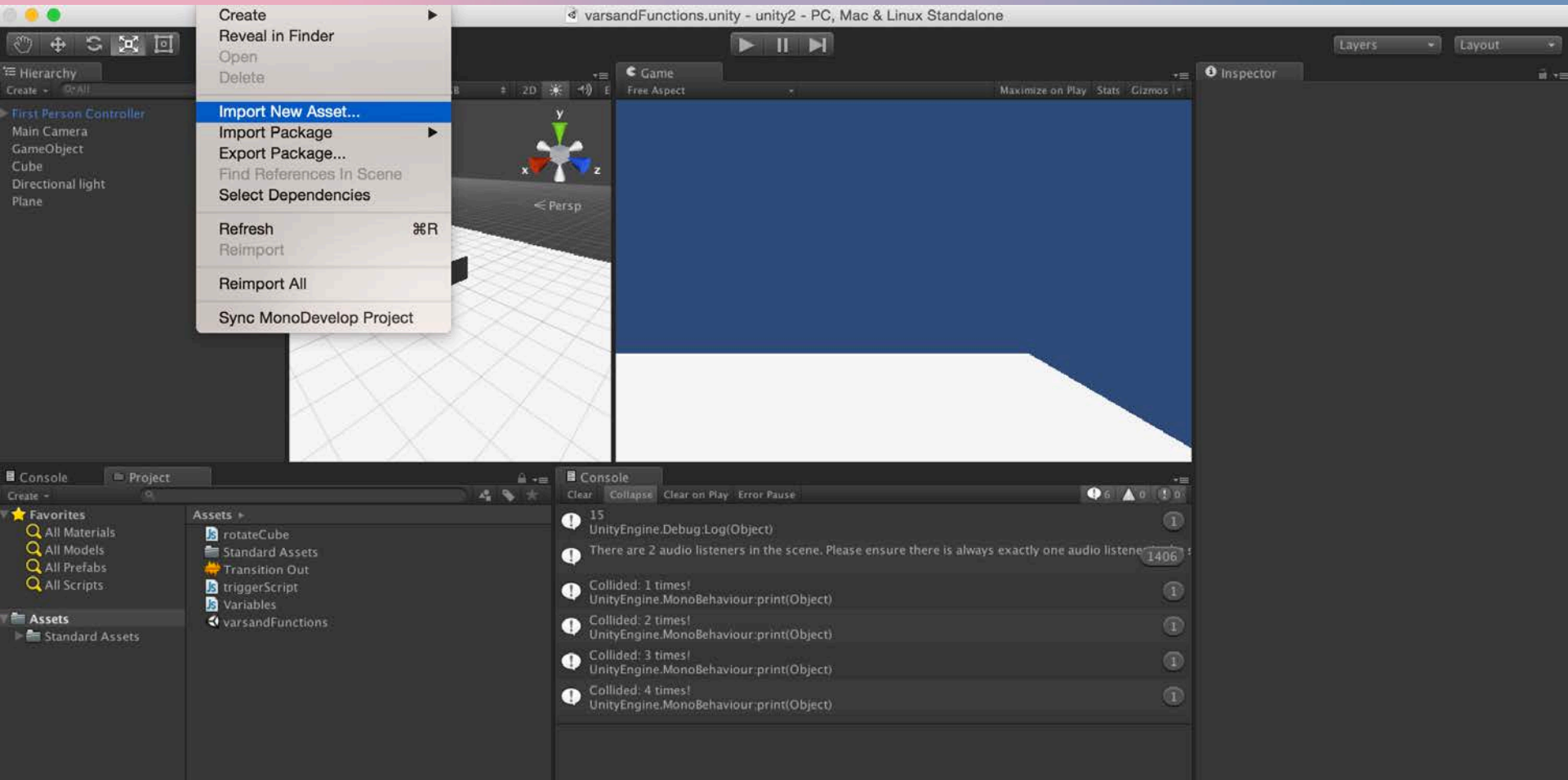
Sounds

- Import new Asset (sound effect/s)
- Add Audio Source to the Cube (Inspector>Add Component >Audio Source)
- Uncheck button “Play On Awake”

Sounds

```
public class ifstatement : MonoBehaviour {  
    public AudioClip mySound;  
    void Update() {  
        if (Input.GetKeyDown(KeyCode.C))  
        {  
            GetComponent<Renderer> ().material.color = Color.cyan;  
        }  
        if (Input.GetKeyDown(KeyCode.M))  
        {  
            GetComponent<Renderer>().material.color = Color.magenta;  
        }  
        if (Input.GetKeyDown(KeyCode.Y))  
        {  
            GetComponent<Renderer>().material.color = Color.yellow;  
            GetComponent<AudioSource>().PlayOneShot(mySound);  
        }  
    }  
}
```

Sounds



Sounds

The image shows the Unity game engine interface. The top-left pane is the Hierarchy, showing a scene with a Main Camera, a Directional light, and a Plane. The top-middle pane is the Scene view, showing a 3D perspective view of a white grid floor with a blue sky and white ground. A black character is standing on the floor, and a green cube is positioned in the center. The top-right pane is the Game view, showing a blue sky and white ground. The bottom-left pane is the Console, showing a list of assets including 'rotateCube', 'Standard Assets', 'Transition Out', 'triggerScript', 'Variables', and 'varsandFunctions'. The bottom-right pane is the Inspector, showing the properties of the selected 'Cube' object. The 'Audio Source' component is selected in the Inspector, and a search bar is visible above it with the text 'audio'. The search results list various audio components, with 'Audio Source' highlighted.

Hierarchy

- First Person Controller
 - Graphics
 - Main Camera
- Main Camera
- GameObject
- Cube
- Directional light
- Plane

Scene

Textured | RGB | 2D | Free Aspect

Position: X 0, Y 0.62, Z 0
Rotation: X 0, Y 0, Z 0
Scale: X 1, Y 1, Z 1

Inspector

Cube | Tag: Untagged | Layer: Default | Static

Transform

Position: X 0, Y 0.62, Z 0
Rotation: X 0, Y 0, Z 0
Scale: X 1, Y 1, Z 1

Cube (Mesh Filter)

Mesh: Cube

Box Collider

Is Trigger:
Material: None (Physic Material)
Center: X 0, Y 0, Z 0
Size: X 1, Y 1, Z 1

Search

audio

- Audio Chorus Filter
- Audio Distortion Filter
- Audio Echo Filter
- Audio High Pass Filter
- Audio Listener
- Audio Low Pass Filter
- Audio Reverb Filter
- Audio Reverb Zone
- Audio Source**
- New Script

Add Component

Sounds

