C#

(C Sharp)

Manipulating the Transform

(Easy Mode)

transform.Rotate

public void **Rotate**(float **xAngle**, float **yAngle**, float **zAngle**, Space **relativeTo** = Space.Self);

Parameters

xAngle	Degrees to rotate around the X axis.
yAngle	Degrees to rotate around the Y axis.
zAngle	Degrees to rotate around the Z axis.
relativeTo	Rotation is local to object or World.

Description

Applies a rotation of `zAngle` degrees around the z axis, `xAngle` degrees around the x axis, and `yAngle` degrees around the y axis (in that order).

If `relativeTo` is not specified or set to Space.Self the rotation is applied around the transform's local axes. If `relativeTo` is set to Space.World the rotation is applied around the world x, y, z axes.

```
using UnityEngine;

public class ExampleClass : MonoBehaviour
{
    void Update()
    {
        // Rotate the object around its local X axis at 1 degree per second
        transform.Rotate(Time.deltaTime, 0, 0);

        // ...also rotate around the World's Y axis
        transform.Rotate(0, Time.deltaTime, 0, Space.World);
    }
}
```

https://docs.unity3d.com/ScriptReference/Transform.Rotate.html

transform.Translate

public void **Translate**(float **x**, float **y**, float **z**, Space **relativeTo** = Space.Self);

Parameters

Description

Moves the transform by x along the x axis, y along the y axis, and z along the z axis.

If `relativeTo` is left out or set to Space.Self the movement is applied relative to the transform's local axes. (the x, y and z axes shown when selecting the object inside the Scene View.) If `relativeTo` is Space.World the movement is applied relative to the world coordinate system.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {
    void Update() {
        transform.Translate(0, 0, Time.deltaTime);
        transform.Translate(0, Time.deltaTime, 0, Space.World);
    }
}
```

https://docs.unity3d.com/ScriptReference/Transform.Translate.html

Events

(A.K.A. Messages)

Messages

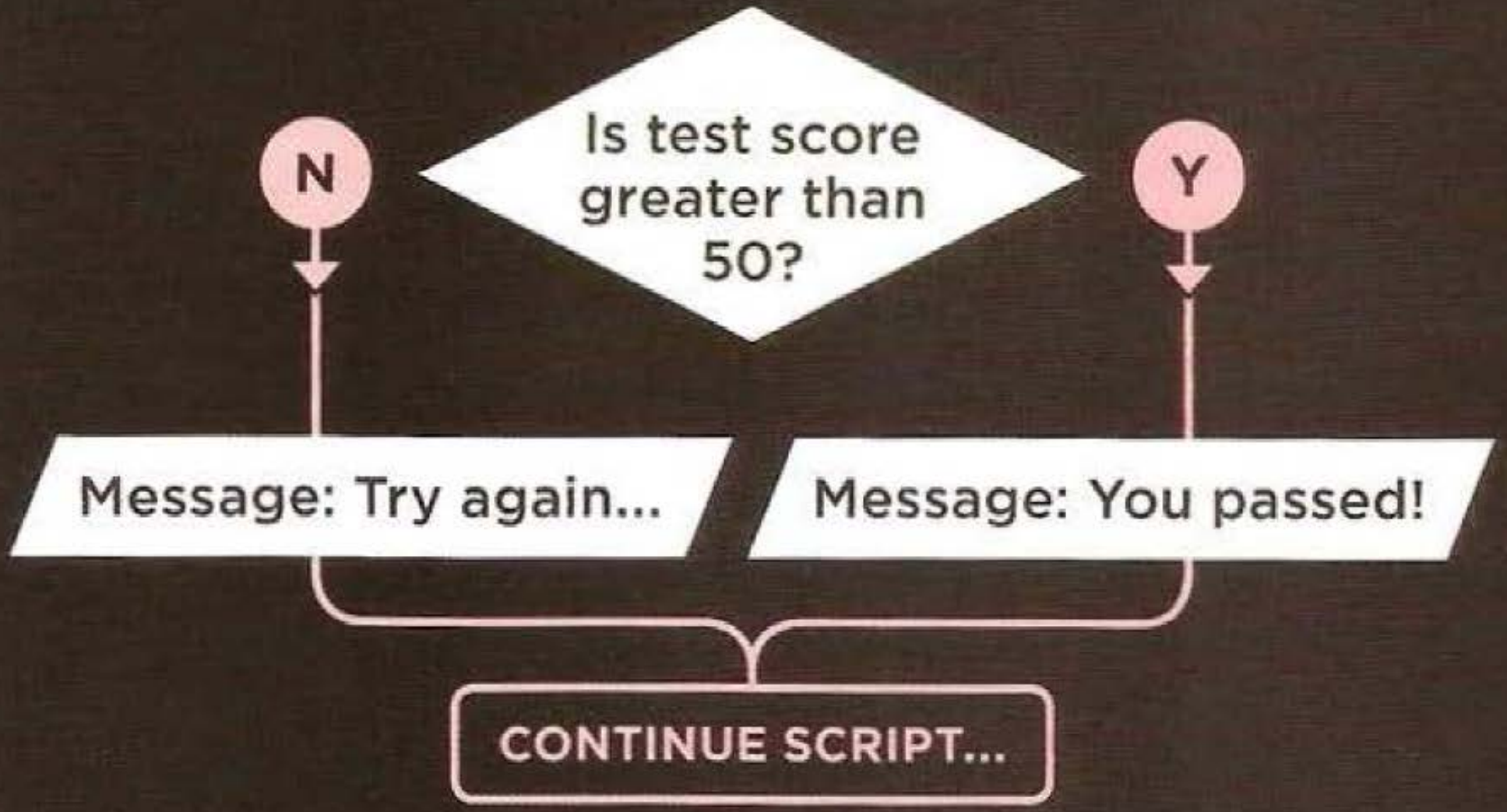
Start	Start is called on the frame when a script is enabled just before any of the Update methods is called the first time.
Update	Update is called every frame, if the MonoBehaviour is enabled.

OnMouseDown	OnMouseDown is called when the user has pressed the mouse button while over the GUIElement or Collider.
OnMouseDrag	OnMouseDrag is called when the user has clicked on a GUIElement or Collider and is still holding down the mouse.
OnMouseEnter	Called when the mouse enters the GUIElement or Collider.
OnMouseExit	Called when the mouse is not any longer over the GUIElement or Collider.
OnMouseOver	Called every frame while the mouse is over the GUIElement or Collider.
OnMouseUp	OnMouseUp is called when the user has released the mouse button.
OnMouseUpAsButton	OnMouseUpAsButton is only called when the mouse is released over the same GUIElement or Collider as it was pressed.

https://docs.unity3d.com/ScriptReference/MonoBehaviour.html

Conditionals

Making Decisions



From JavaScript & jQuery

```
float score = 100f;
```

CONDITION

```
if (score >= 50) {
    Debug.Log("The score was a passing grade :)");
}
```

RESULT

Comparison Operators

>

>=

<

<=

==

!=

Boolean Variables

```
bool isRaining = true;
bool hasPressedKey = false;
```

Boolean Variables

```
float score = 100f;
bool isPassing = score >= 50;
```

```
if (score >= 50) {
    Debug.Log("The score was a passing grade :)");
} else {
    Debug.Log("The score was a failure :(");
}
```

```
if (score >= 95) {
    Debug.Log("A+ bro.");
} else if (score >= 50) {
    Debug.Log("The score was a passing grade :)");
} else {
    Debug.Log("The score was a failure :(");
}
```

```
// Create a function that takes a (float) number and returns a string with
// the corresponding letter grade. Use this grade scale:
//
//  A = 90 to 100
//  B = 80 up to (but not including) 90
//  C = 70 up to (but not including) 80
//  D = 60 up to (but not including) 70
//  F = Anything below a 60
```

```
string determineGrade(float score) {
    if (score >= 90) {
        return "A";
    } else if (score >= 80) {
        return "B";
    } else if (score >= 70) {
        return "C";
    } else if (score >= 60) {
        return "D";
    } else {
        return "F";
    }
}
```

```
string determineGrade(float score) {
    string grade;
    if (score >= 90) {
        grade = "A";
    } else if (score >= 80) {
        grade = "B";
    } else if (score >= 70) {
        grade = "C";
    } else if (score >= 60) {
        grade = "D";
    } else {
        grade = "F";
    }
    return grade;
}
```

Getting Input

(Quick Way)

Input.GetKey

public static bool **GetKey**(string **name**);

Parameters

Description

Returns true while the user holds down the key identified by `name`. Think auto fire.

For the list of key identifiers see Input Manager. When dealing with input it is recommended to use Input.GetAxis and Input.GetButton instead since it allows end-users to configure the keys.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {
    void Update() {
        if (Input.GetKey("up"))
            print("up arrow key is held down");

        if (Input.GetKey("down"))
            print("down arrow key is held down");

    }
}
```

https://docs.unity3d.com/ScriptReference/Input.GetKey.html

Input.GetKeyDown

public static bool **GetKeyDown**(string **name**);

Parameters

Description

Returns true during the frame the user starts pressing down the key identified by `name`.

You need to call this function from the Update function, since the state gets reset each frame. It will not return true until the user has released the key and pressed it again.

For the list of key identifiers see Input Manager. When dealing with input it is recommended to use Input.GetAxis and Input.GetButton instead since it allows end-users to configure the keys.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {
    void Update() {
        if (Input.GetKeyDown("space"))
            print("space key was pressed");

    }
}
```

https://docs.unity3d.com/ScriptReference/Input.GetKeyDown.html

Input.GetKeyUp

public static bool **GetKeyUp**(string **name**);

Parameters

Description

Returns true during the frame the user releases the key identified by `name`.

You need to call this function from the Update function, since the state gets reset each frame. It will not return true until the user has pressed the key and released it again.

For the list of key identifiers see Input Manager. When dealing with input it is recommended to use Input.GetAxis and Input.GetButton instead since it allows end-users to configure the keys.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {
    void Update() {
        if (Input.GetKeyUp("space"))
            print("space key was released");

    }
}
```

https://docs.unity3d.com/ScriptReference/Input.GetKeyUp.html

Keys

The names of keys follow this convention:

- Normal keys: “a”, “b”, “c” ...
- Number keys: “1”, “2”, “3”, ...
- Arrow keys: “up”, “down”, “left”, “right”
- Keypad keys: “[1]”, “[2]”, “[3]”, “[+]”, “[equals]”
- Modifier keys: “right shift”, “left shift”, “right ctrl”, “left ctrl”, “right alt”, “left alt”, “right cmd”, “left cmd”
- Mouse Buttons: “mouse 0”, “mouse 1”, “mouse 2”, ...
- Joystick Buttons (from any joystick): “joystick button 0”, “joystick button 1”, “joystick button 2”, ...
- Joystick Buttons (from a specific joystick): “joystick 1 button 0”, “joystick 1 button 1”, “joystick 2 button 0”, ...
- Special keys: “backspace”, “tab”, “return”, “escape”, “space”, “delete”, “enter”, “insert”, “home”, “end”, “page up”, “page down”
- Function keys: “f1”, “f2”, “f3”, ...

The names used to identify the keys are the same in the scripting interface and the Inspector.

```
value = Input.GetKey ("a");
```

https://docs.unity3d.com/Manual/ConventionalGameInput.html

Input.GetAxis

public static float **GetAxis**(string **axisName**);

Parameters

Description

Returns the value of the virtual axis identified by `axisName`.

The value will be in the range -1...1 for keyboard and joystick input. If the axis is setup to be delta mouse movement, the mouse delta is multiplied by the axis sensitivity and the range is not -1...1.

This is frame-rate independent; you do not need to be concerned about varying frame-rates when using this value.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {
    public float horizontalSpeed = 2.0F;
    public float verticalSpeed = 2.0F;
    void Update() {
        float h = horizontalSpeed * Input.GetAxis("Mouse X");
        float v = verticalSpeed * Input.GetAxis("Mouse Y");
        transform.Rotate(v, h, 0);
    }
}
```

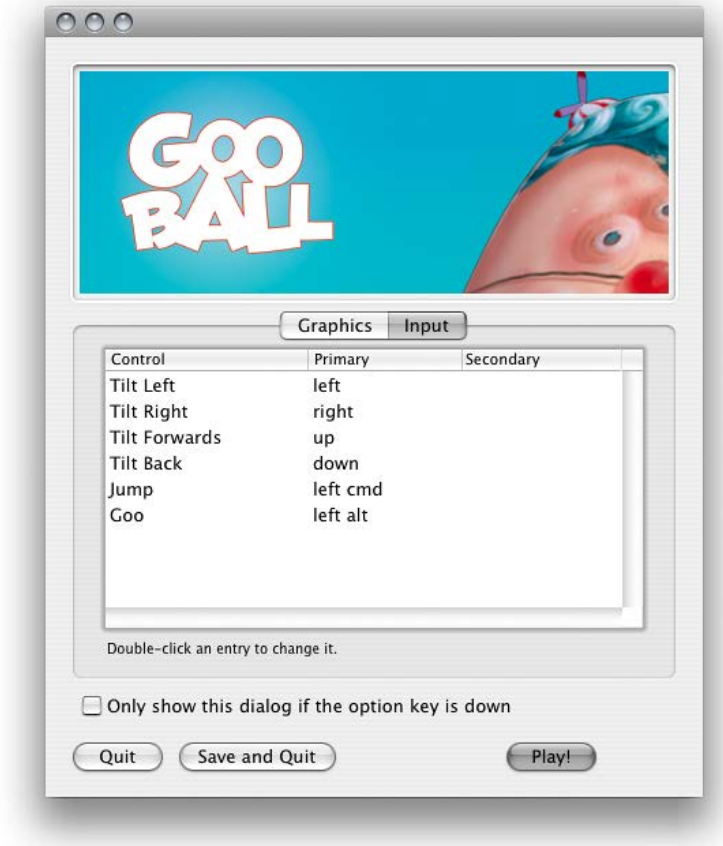
https://docs.unity3d.com/ScriptReference/Input.GetAxis.html

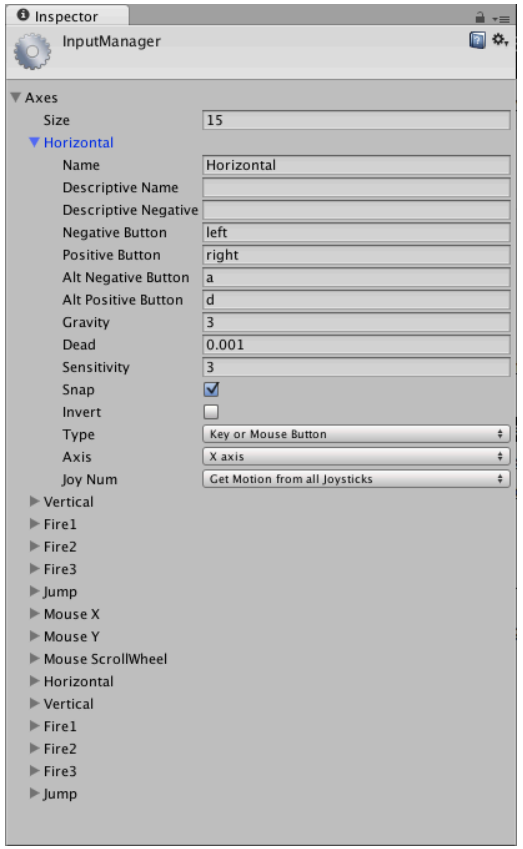
More Mouse Inputs

- Input.GetMouseButton
- Input.GetMouseButtonDown
- Input.GetMouseButtonUp

Customizable Input

See https://docs.unity3d.com/Manual/Input.html





Euler vs Quaternions

Euler Rotation

```
float horizontalMovement = Input.GetAxis("Mouse X");
float verticalMovement = Input.GetAxis("Mouse Y");

// Wrong way to rotate along two axes! Don't do this.
transform.Rotate(0, horizontalMovement, 0);
transform.Rotate(-verticalMovement, 0, 0);
```


Gimbal Lock
in 30 seconds

Quaternion.Euler

public static Quaternion **Euler**(float **x**, float **y**, float **z**);

Parameters

Description

Returns a rotation that rotates z degrees around the z axis, x degrees around the x axis, and y degrees around the y axis (in that order).

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {
    public Quaternion rotation = Quaternion.Euler(0, 30, 0);
}
```

https://docs.unity3d.com/ScriptReference/Quaternion.Euler.html

Quaternion Rotation

```
// Rotating with quaternions - much better!
transform.localRotation = Quaternion.Euler(45f, 20f, 0f);
```

Classes and Instances

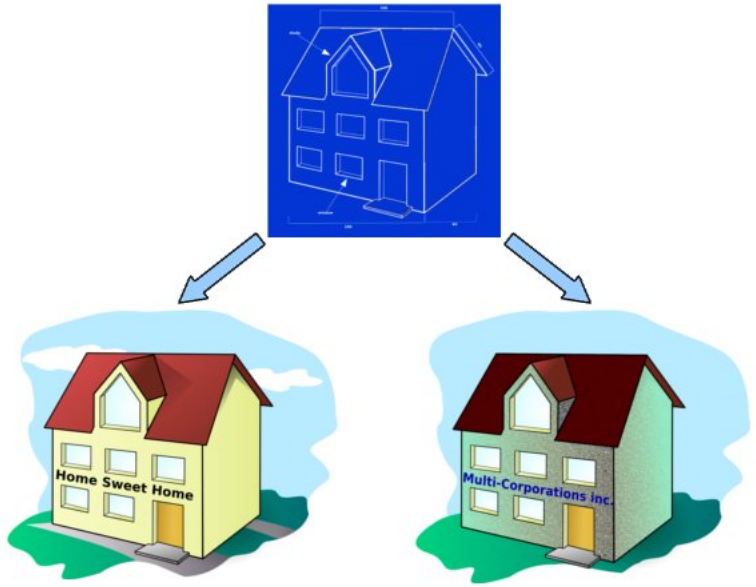
(OOP)

Classes & Instances

- Encapsulation: organize variables and functions together
- Code reuse

Analogy Time

- Blueprint -> House
- Cookie Cutter -> Cookie
- Person -> Bob



http://processing.lyndondaniels.com/53blueprint.php

Fields

Class

ACCESS MODIFIER

CLASS NAME

```
public class Enemy {
    // Fields and methods go inside brackets
}
```

Fields

```
public class Enemy {
    // Fields
    public string Name;
}
```

ACCESS MODIFIER

VARIABLE TYPE

VARIABLE NAME

INSTANCE

```
public class ClassDemo : MonoBehaviour {

    void Start () {
        Enemy enemy1 = new Enemy();
    }

}

public class Enemy {
    // Fields
    public string Name;
}
```

CLASS

VARIABLE NAME

CONSTRUCTOR

```
Enemy enemy1 = new Enemy();
```

VARIABLE TYPE

KEYWORD

INSTANCE

CLASS

```
public class ClassDemo : MonoBehaviour {

    void Start () {
        Enemy enemy1 = new Enemy();
        enemy1.Name = "Carl The Goblin";
        Debug.Log("Monster 1 is: " + enemy1.Name);
    }

}

public class Enemy {
    // Fields
    public string Name;
}
```

DOT OPERATOR

```
enemy1.Name = "Carl the Goblin";
```

INSTANCE

FIELD

INSTANCE

INSTANCE

CLASS

```
public class ClassDemo : MonoBehaviour {

    void Start () {
        Enemy enemy1 = new Enemy();
        enemy1.Name = "Carl The Goblin";
        Debug.Log("Monster 1 is: " + enemy1.Name);

        Enemy enemy2 = new Enemy();
        enemy2.Name = "Radcliff";
        Debug.Log("Monster 2 is: " + enemy2.Name);
    }

}

public class Enemy {
    // Fields
    public string Name;
}
```

Constructors

```
public class Enemy {
    // Fields
    public string Name;

    // Constructor
    public Enemy(string name) {
        Name = name;
    }
}
```

```
public class ClassDemo : MonoBehaviour {
    void Start () {
        Enemy enemy1 = new Enemy("Carl The Goblin");
        Debug.Log("Enemy 1 is: " + enemy1.Name);
    }
}

public class Enemy {
    // Fields
    public string Name;

    // Constructor
    public Enemy(string name) {
        Name = name;
    }
}
```

Methods

```
public class Enemy {
    // Fields
    public string Name;

    // Constructor
    public Enemy(string name) {
        Name = name;
    }

    // Methods
    public void Speak() {
        Debug.Log("Hello, I am " + Name + ".");
    }
}
```

```
Enemy enemy1 = new Enemy("Carl The Goblin");
enemy1.Speak();
```

More Methods

```
public class Enemy {
    // Fields
    public string Name;
    private int Health;

    // Constructor
    public Enemy(string name, int health) {
        Name = name;
        Health = health;
    }

    // Methods
    public void Speak() {
        Debug.Log("Hello, I am " + Name + ".");
    }
    public void TakeDamage(int damage) {
        Health = Health - damage;
    }
    public void SayHealth() {
        if (Health < 0) {
            Debug.Log(Name + ": I am dead :(");
        } else {
            Debug.Log(Name + ": I have " + Health + " health");
        }
    }
}
```

```
// Create a Rectangle class:
//  - It should have public fields for Width & Height
//  - It should have a constructor that can initialize the Width and Height
//  - It should have a method GetArea that returns a float
//  - It should have a method GetPerimeter that returns a float
//
// Test it by creating a couple Rectangle instances and calculating the area
// and perimeter
```

Scripts are Classes

```
public class Rotator : MonoBehaviour {

    // Use this for initialization
    void Start () {

    }

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

    }
}
```

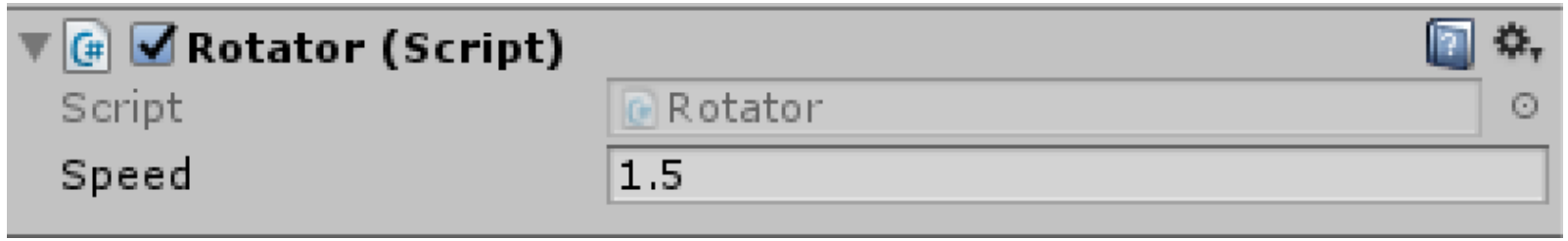
```
public class Rotator : MonoBehaviour {

    public float speed;

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {
        float rotationAmount = speed * Time.deltaTime;
        transform.Rotate(rotationAmount, 0f, 0f);
    }
}
```




```
public class Rotator : MonoBehaviour {

    public float speed = 10;

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {
        float rotationAmount = speed * Time.deltaTime;
        transform.Rotate(rotationAmount, 0f, 0f);
    }
}
```

Accessing Components

Via Inspector

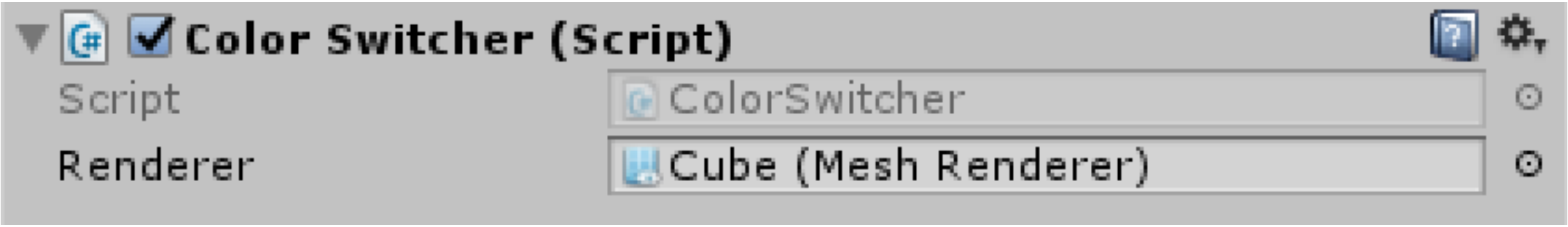
```
public class ColorSwitcher : MonoBehaviour {

    public Renderer renderer;

    // Use this for initialization
    void Start () {

    }

}
```



Via Scripting

```
public class ColorSwitcher : MonoBehaviour {

    private Renderer renderer;

    // Use this for initialization
    void Start () {
        renderer = GetComponent<Renderer>();
    }

}
```

Generic Method

```
renderer = GetComponent<Renderer>();
```

TYPE OF COMPONENT