

NEW TECHNOLOGY IN IT APPLICATION DEVELOPMENT

Scripts in Unity

frames

- A frame is a term inherited from animation.
- 24, 30, and 60 frames per second (FPS).
 - Example, it's running at 60 FPS that means there are 60 new images in a second presented to the player, called frame rate.
- frame interval is the time that happens between each frame
 - Example of 60 FPS, 60 would be the frame rate, and frame interval would be $1/60 \times 1000 = 16.67$ milliseconds.

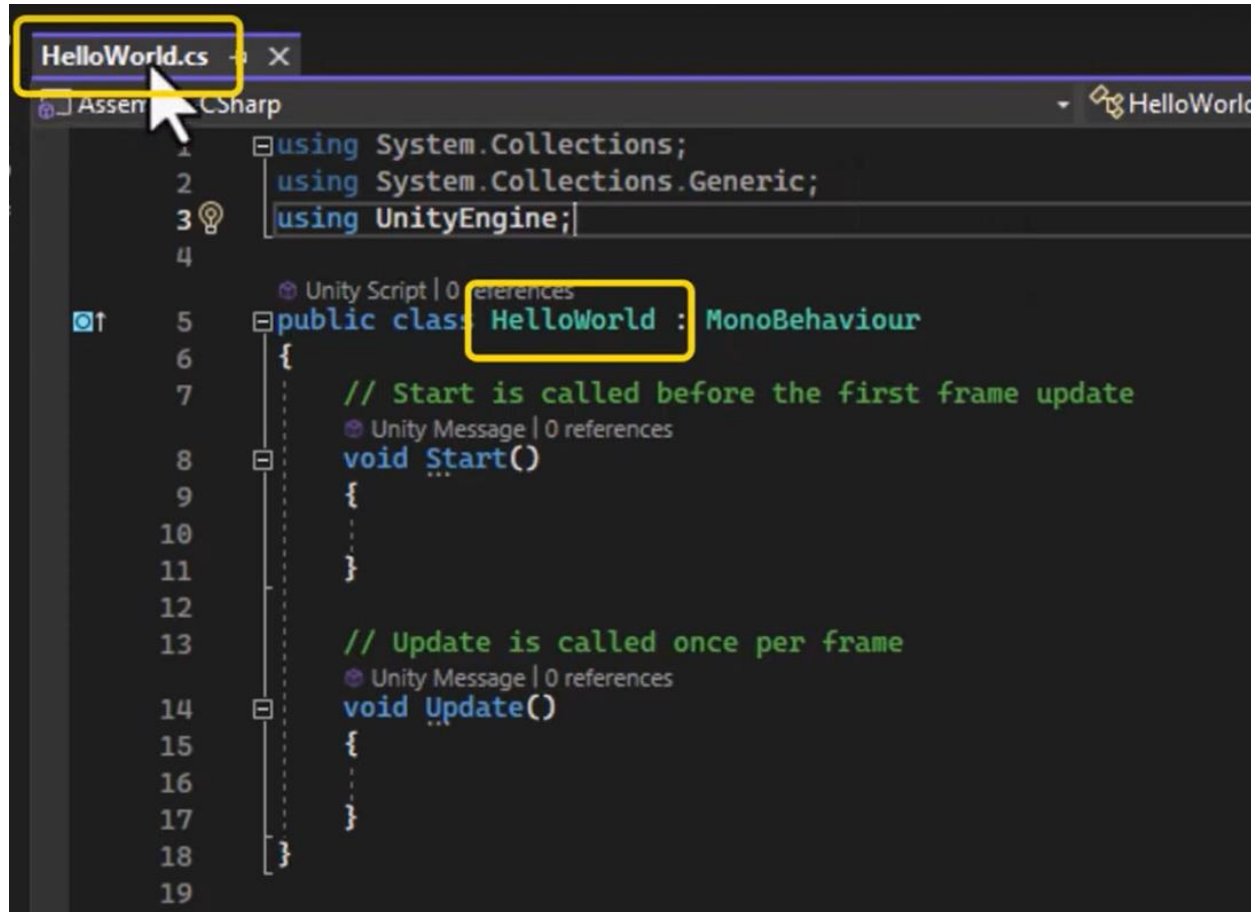
frame in Unity

- A frame is considered a rendered image presented to the player's screen
- Care about frame's property `Time.deltaTime` that calculates time between frames
- If having heavy processing during gameplay, time between frames slows down, this drops the frame rate and gives the sensation to the player that the game is running slowly.

Scripts in Unity

- Scripts as Behaviour Components
- As the other components of Unity, they can be applied objects.
- Using “Add Component” of Object to attach the script

C# Script



The screenshot shows a C# script named `HelloWorld.cs` in a code editor. The script is a Unity script that inherits from `MonoBehaviour`. It includes the following code:

```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class HelloWorld : MonoBehaviour
6 {
7     // Start is called before the first frame update
8     void Start()
9     {
10
11     }
12
13     // Update is called once per frame
14     void Update()
15     {
16
17     }
18 }
19
```

The file name `HelloWorld.cs` in the tab and the class name `HelloWorld` are highlighted with yellow boxes. A mouse cursor is pointing at the `using UnityEngine;` line.

HelloWorld

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

public class HelloWorld : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {

    }

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

    }
}
```

Run one time when this class is launched

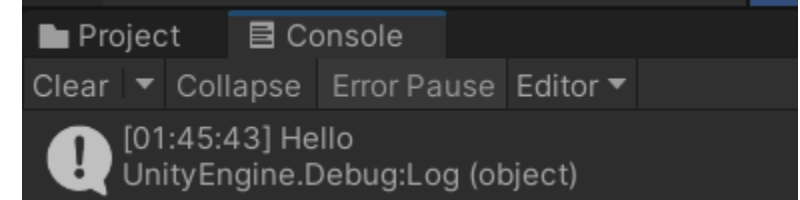
Run once every frame as the project runs that means 30 times a second

HelloWorld

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

public class HelloWorld : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {
        Debug.Log("Hello");
    }

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

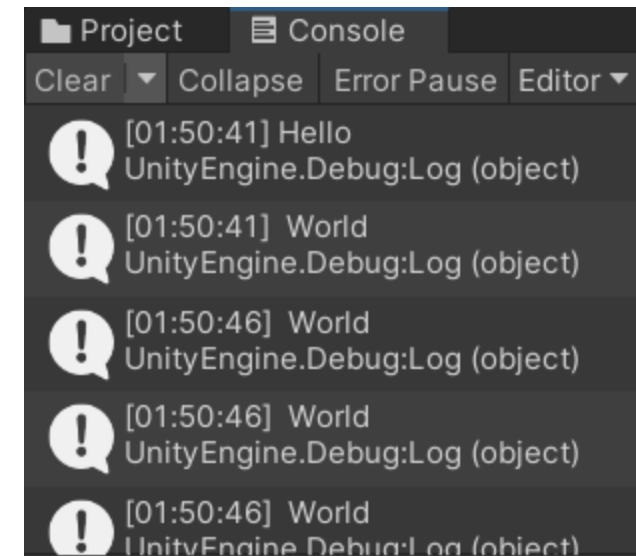


HelloWorld

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

public class HelloWorld : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {
        Debug.Log("Hello");
    }

    // Update is called once per frame
    void Update()
    {
        Debug.Log(" World");
    }
}
```



Hello Program

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

public class HelloWorld : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {

    }

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

    }
}
```

Data Types

Value type

- Int
- Float
- Double
- Bool
- Char
- Struct
 - Vector
 - Quaternion

Reference type

- Classes
 - Transform
 - GameObject

Data Types

```
using UnityEngine;
using System.Collections;

public class DatatypeScript : MonoBehaviour
{
    void Start ()
    {
        //Value type variable
        Vector3 pos = transform.position;
        pos = new Vector3(0, 2, 0);

        //Reference type variable
        Transform tran = transform;
        tran.position = new Vector3(0, 2, 0);
    }
}
```

Variables

- A variable is a named place in the memory where a programmer can store data and later retrieve the data using the variable “name”
- Programmers get to choose the names of the variables
- You can change the contents of a variable in a later statement

x 12.2

y 14

x 12.2

y 14

Functions

- a function is some reusable code that takes **arguments**(s) as input, does some computation, and then returns a result or results

Variables and Functions

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

public class EmptyBV : MonoBehaviour
{
    int x = 5;
    void Start() {
        x = 70;
        Debug.Log(x);
    }
    void Update() {
    }
}
```

`int`: integer type

`x` variable

`void`: function don't
return value

Variables and Functions

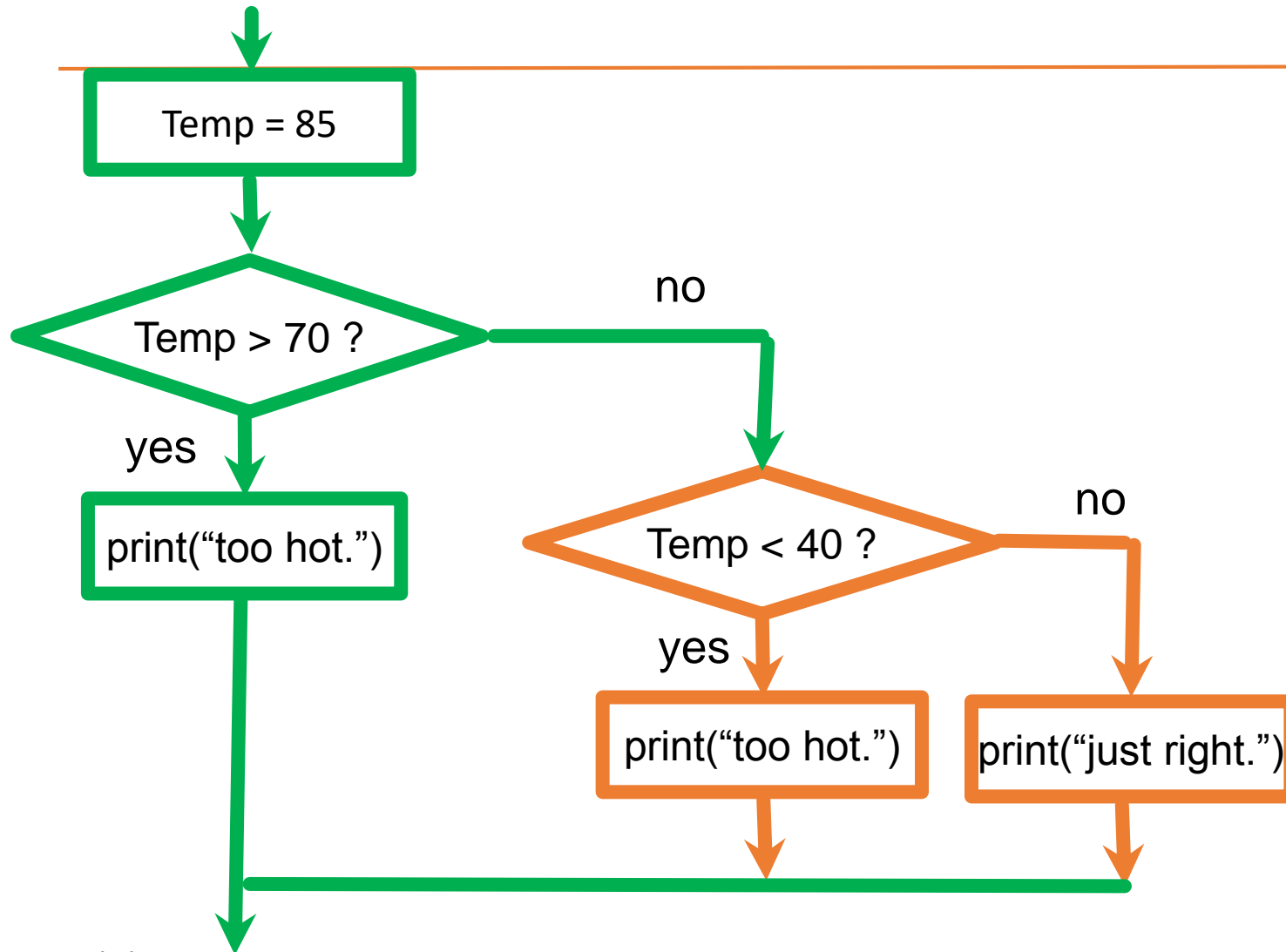
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class EmptyBV : MonoBehaviour
{
    int x = 5;
    int MultiplyByTwo(int num) {
        int res = x*2;
        return res;
    }
    void Start() {
        x = MultiplyByTwo (x);
        Debug.Log(x);
    }
}
```

MultiplyByTwo function
return value, its type is `int`

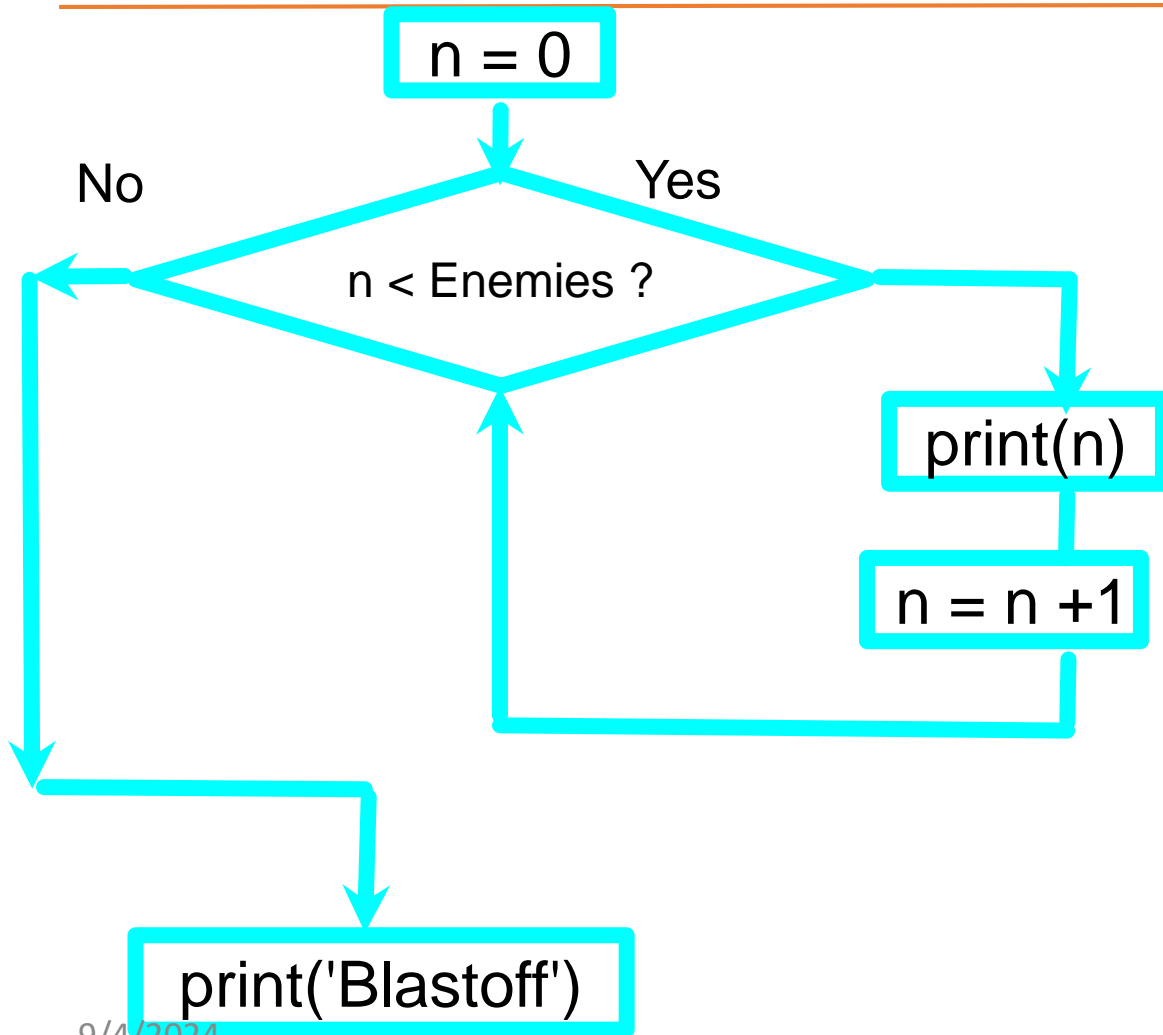
Call MultiplyByTwo

Definition if



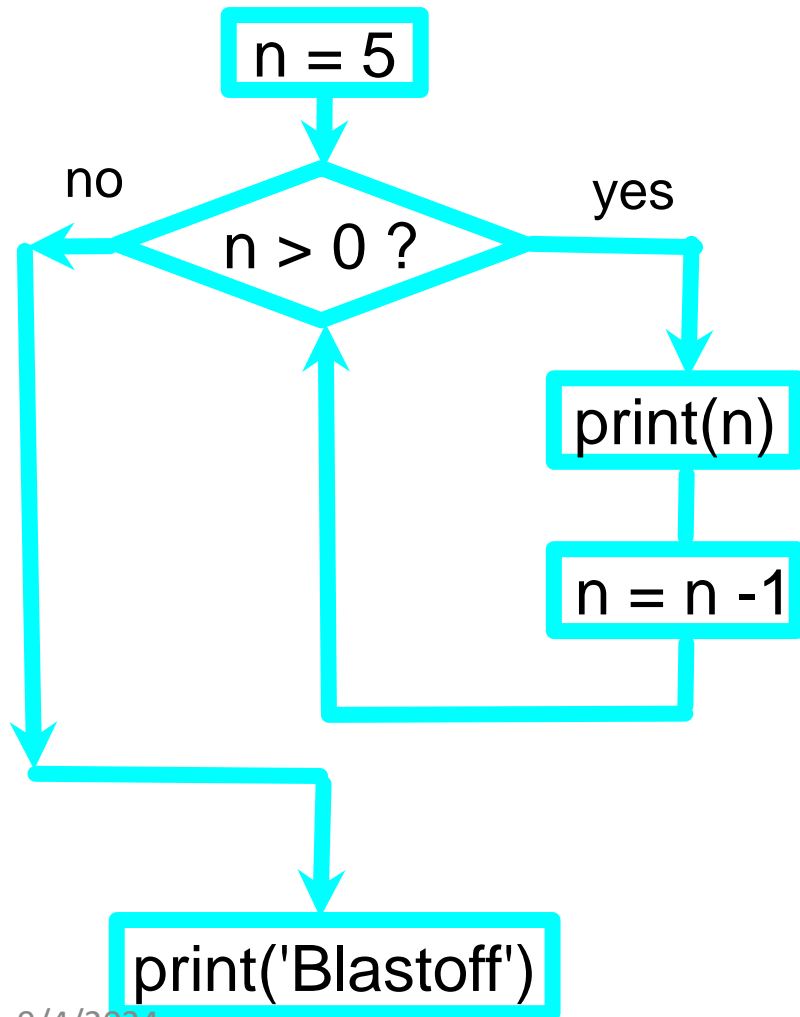
```
Temp = 85;  
if(Temp > 70)  
{  
    print("too hot.");  
}  
else if(Temp < Temp)  
{  
    print("too cold.");  
}  
else  
{  
    print("just right.");  
}
```


Definition for



```
void Start ()  
{  
    for(int n = 0; n < Enemies; n++)  
    {  
        Debug.Log("Creating enemy  number: " + n);  
    }  
    Debug.Log("Blastoff");  
}
```

Definition while



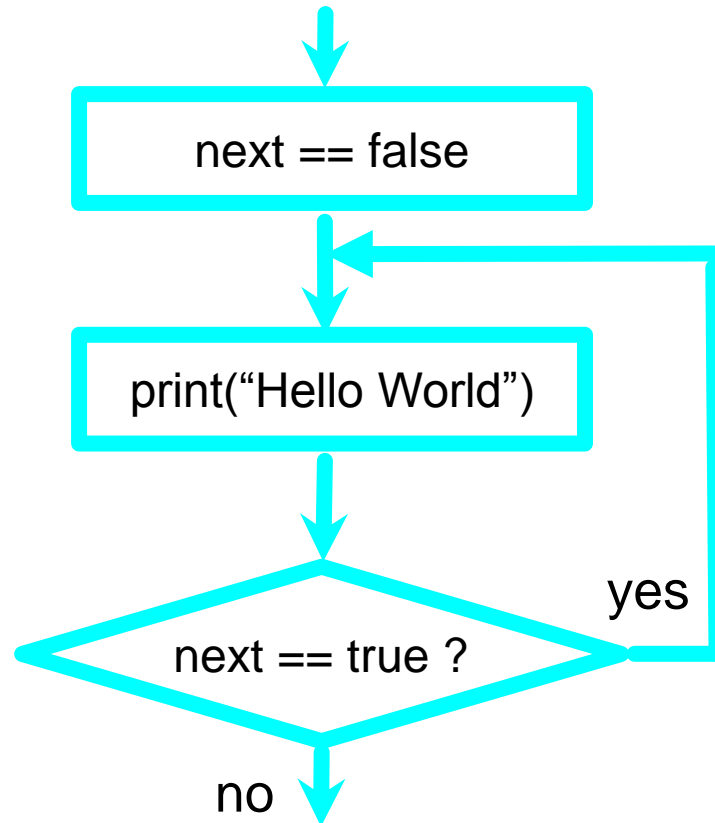
Program:

```
while(n > 0)
{
    print (n);
    n = n - 1;
}
print("Blastoff!");
print(n);
```

Output:

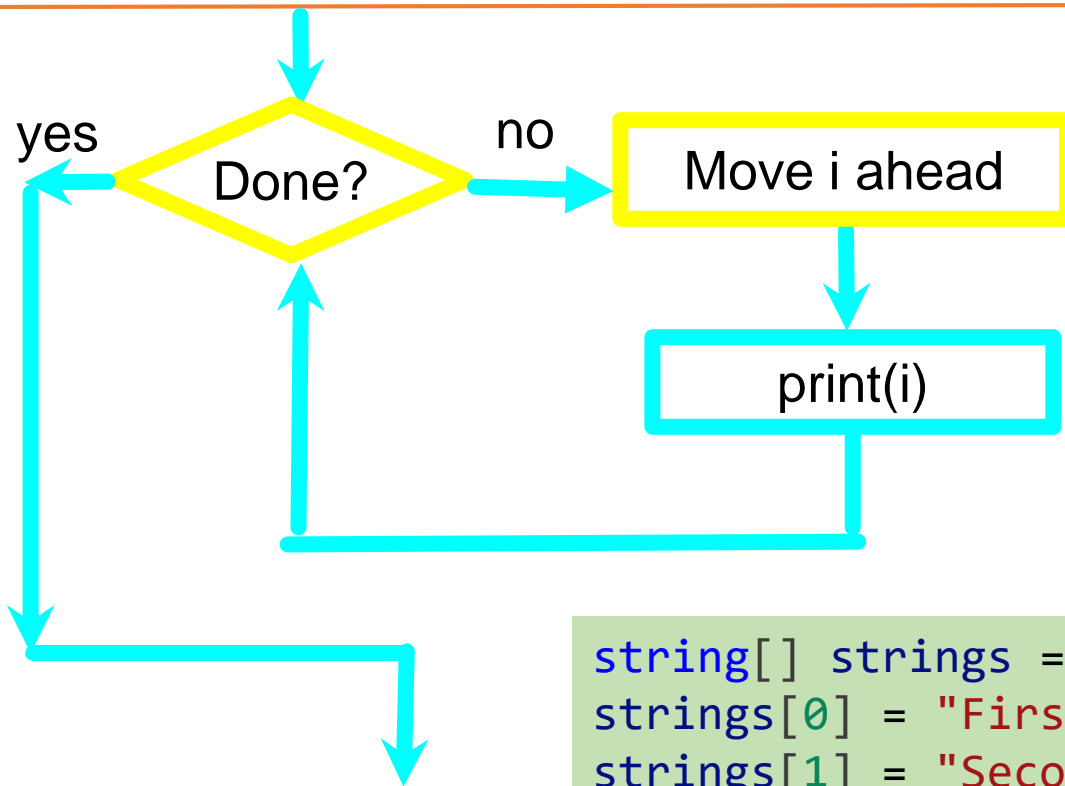
5
4
3
2
1
Blastoff!
0

Definition DoWhile

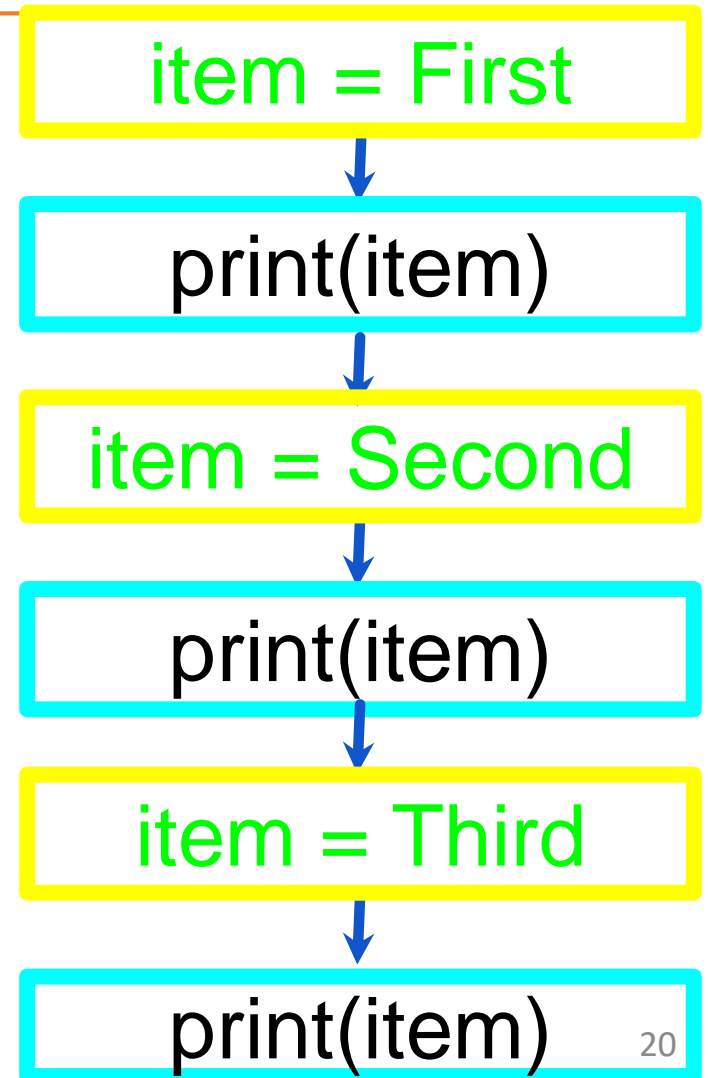


```
bool next = false;  
do  
{  
    print ("Hello World");  
}while(next == true);
```

Definition foreach



```
string[] strings = new string[3];  
strings[0] = "First";  
strings[1] = "Second";  
strings[2] = "Third";  
foreach(string item in strings)  
    print (item);
```



Scope and Access Modifiers

```
public class ScopeModifier : MonoBehaviour
```

```
{
```

```
    public int alpha = 5;
```

```
    private int beta = 0;
```

```
    private int gamma = 5;
```

alpha, beta,
gamma scope
within **class**

Access Modifiers

```
void Example(int pens, int crayons)
```

```
{
```

```
    int answer;
```

```
    answer = pens*crayons*alpha;
```

```
    Debug.Log(answer);
```

```
}
```

pens, crayons,
answer scope
within **function**

Alpha can be
seen from
outside of class

```
// Update is called once per frame
```

```
void Update()
```

```
{
```

```
    Debug.Log("Alpha is set to: " + alpha);
```

```
}
```

```
}
```

Scope and Access Modifiers

- Alpha will not be overridden when program start

```
void Start()  
{  
    alpha = 79;  
}
```

```
public class AnotherClass : MonoBehaviour
{
    public int apples;
    public int bananas;

    private int stapler;
    private int sellotape;

    public void FruitMachine (int a, int b)
    {
        int answer;
        answer = a + b;
        Debug.Log("Fruit total: " + answer);
    }

    private void OfficeSort (int a, int b)
    {
        int answer;
        answer = a + b;
        Debug.Log("Office Supplies total: " + answer);
    }
}
```

```
void Start()
{
    alpha = 79;
    myOtherClass = new
    AnotherClass();
    myOtherClass.
}
```

Awake and Start

- Use Awake to initialize variables or states before the application starts
- Unity calls Awake only once during the lifetime of the script instance.
 - A script's lifetime lasts until the Scene that contains it is unloaded
- If the Scene is loaded again, Unity loads the script instance again and calls Awake again. Awake is called once for each instance.
- Awake is always called before any Start function
- Use Awake to set up references between scripts, and use Start, which is called after all Awake calls are finished, to pass any information back and forth

Update and FixedUpdate

Update

- Called every frame
- Used for regular update such as:
 - Moving non-physics objects
 - Simple Timers
 - Receiving input
- Update interval times vary

FixedUpdate

- Called every physics step
- FixedUpdate intervals are consistent
- Used for regular updates such as:
 - Adjusting physics (Rigidbody) objects

Update and FixedUpdate

Update

```
void Update ()  
{  
    Debug.Log("Update time  
:" + Time.deltaTime);  
}
```

FixedUpdate

```
void FixedUpdate ()  
{  
    Debug.Log("FixedUpdate  
time :" +  
Time.deltaTime);  
}
```

Unity's order of event functions visualised

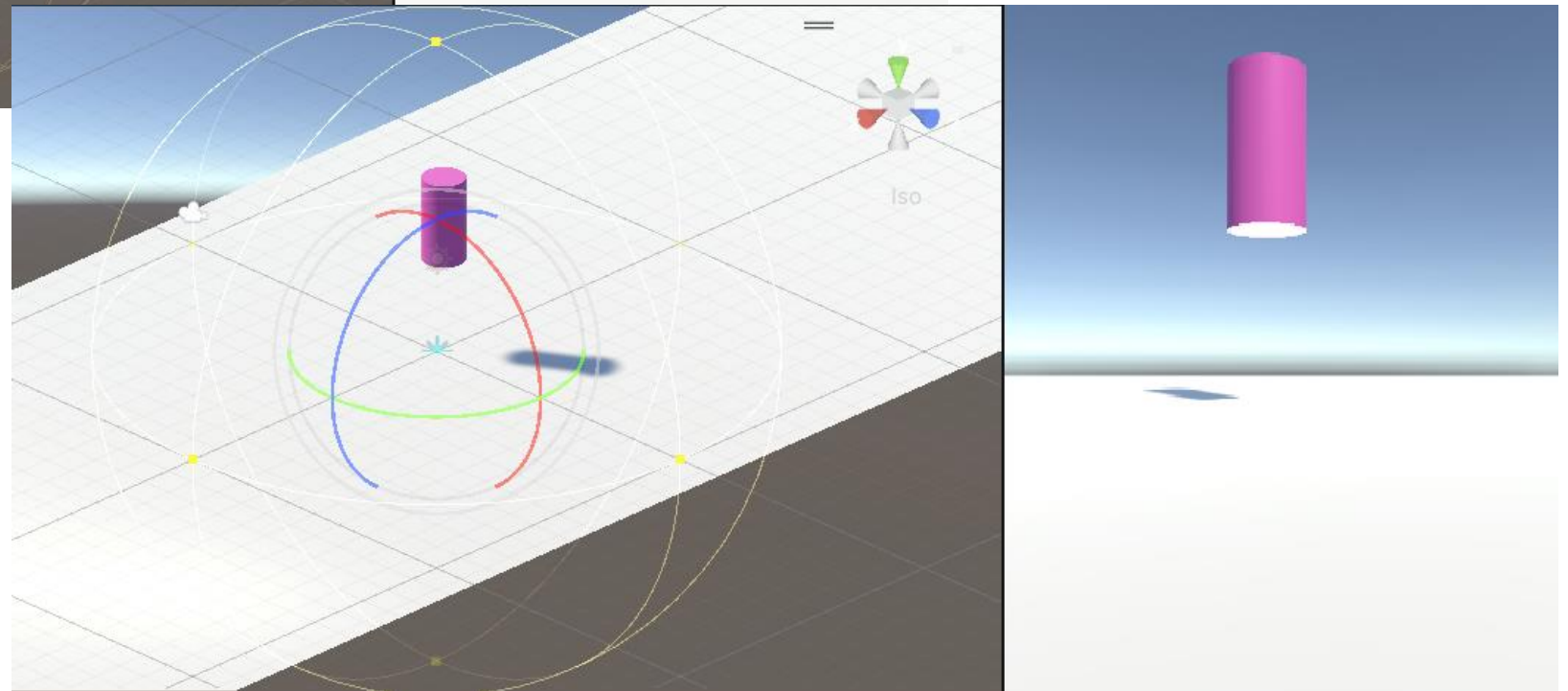
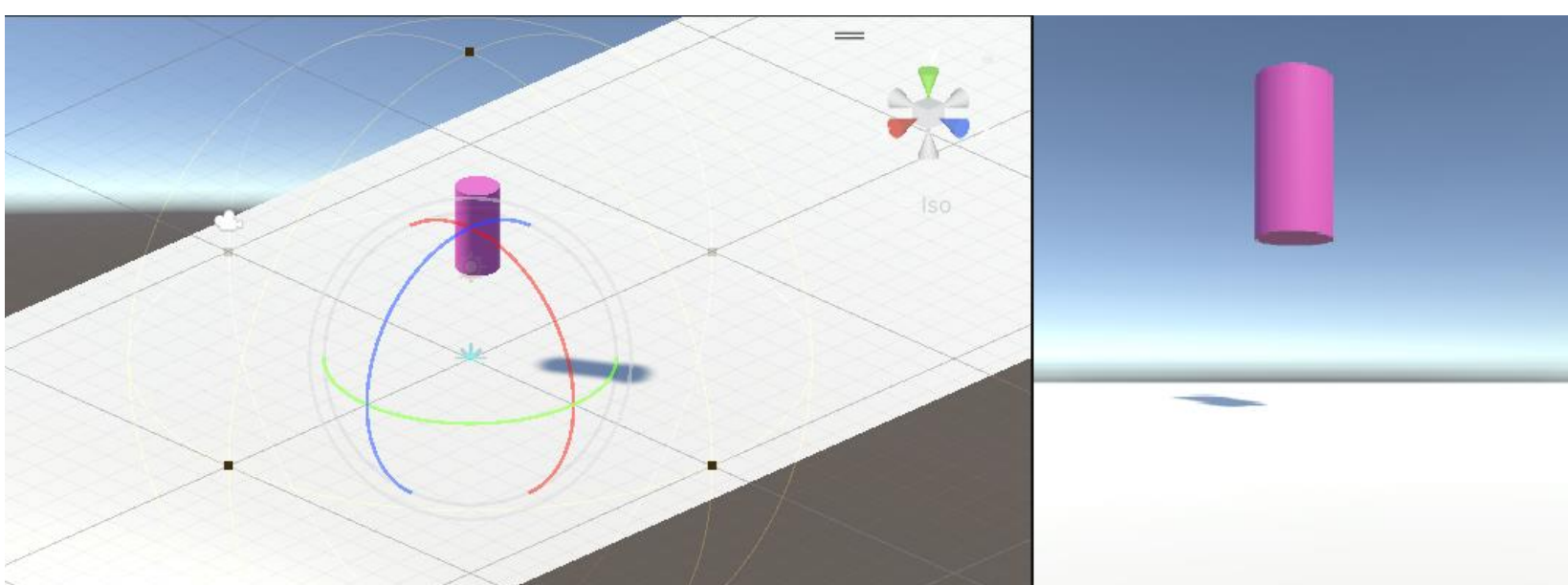


Source: <https://gamedevbeginner.com/start-vs-awake-in-unity/>

Enabling and Disabling Components

```
public class EnableComponents : MonoBehaviour
{
    private Light myLight;
    void Start()
    {
        myLight = GetComponent<Light>();
    }

    void Update()
    {
        if(Input.GetKeyUp(KeyCode.Space))
        {
            myLight.enabled = !myLight.enabled;
        }
    }
}
```



Translate and Rotate

- public void Translate([Vector3](#) translation);
- public void Translate([Vector3](#) translation, [Space](#) relativeTo = Space.Self);
- relativeTo = Space.Self: The movement is applied relative to the transform's local axes.
- relativeTo = Space.World: The movement is applied relative to the world coordinate system

Translate and Rotate

```
transform.Translate(0, 0, 1);  
transform.Translate(new Vector3(0,0,1));
```

Move by one along z axis,
every frame

```
transform.Translate(0, 0, Time.deltaTime);  
  
public float speed = 10f;  
transform.Translate(Vector3.forward*speed*Time.deltaTime);
```

Move by one along z axis,
unit/second

```
transform.Translate(0, Time.deltaTime, 0, Space.World);  
  
public float movespeed = 10f;  
transform.Translate(-Vector3.up*movespeed*Time.deltaTime,  
Space.World);
```

Move by one along y axis,
unit/second

Translate and Rotate

```
public float turn = 10f;  
transform.Rotate(Vector3.up, -turn*Time.deltaTime);
```

```
public float turn = 10f;  
transform.Rotate(Vector3.up, turn*Time.deltaTime);
```


Look At

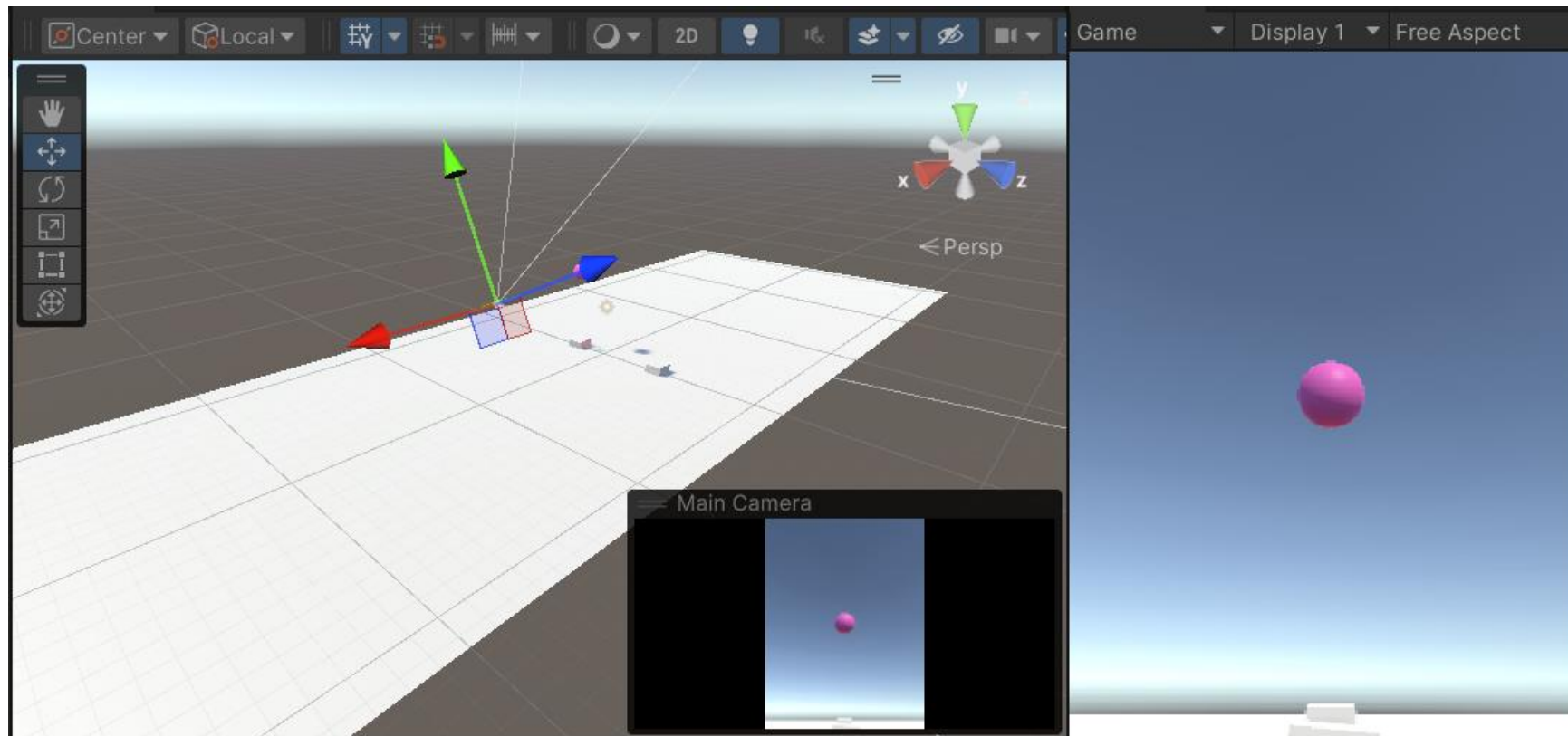
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CameraLookAt : MonoBehaviour
{
    public Transform target;

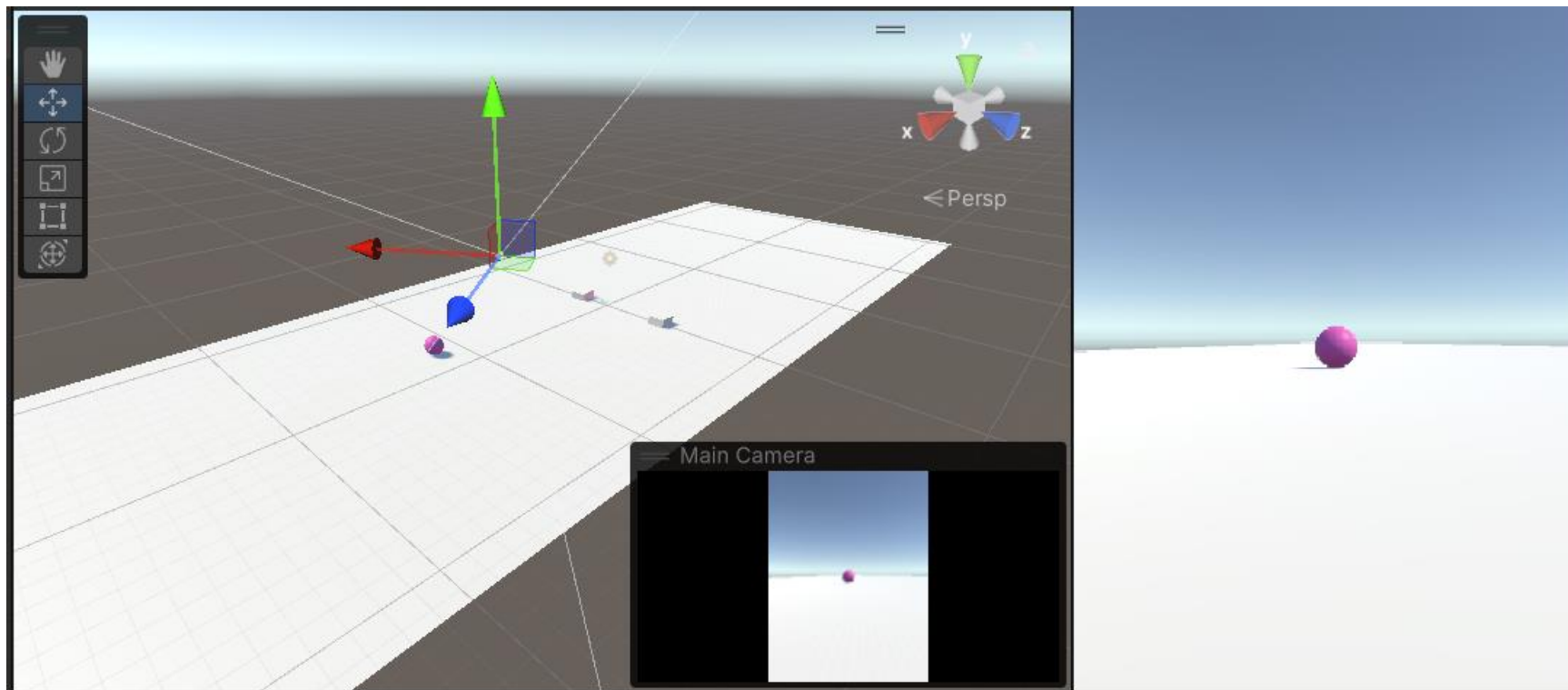
    void Update()
    {
        transform.LookAt(target);
    }
}
```

Target is observed object

Look At



Look At



Linear Interpolation

- Linearly interpolating is finding a value that is some percentage between two given values. `Mathf.Lerp(from, to, percent)`

```
// In this case, result = 4  
float result = Mathf.Lerp (3f, 5f, 0.5f);
```

```
Vector3 from = new Vector3 (1f, 2f, 3f);  
Vector3 to = new Vector3 (5f, 6f, 7f);
```

```
// Here result = (4, 5, 6)  
Vector3 result = Vector3.Lerp (from, to, 0.75f);
```

Linear Interpolation

- In the Color struct, colours are represented by 4 floats representing red, blue, green and alpha.

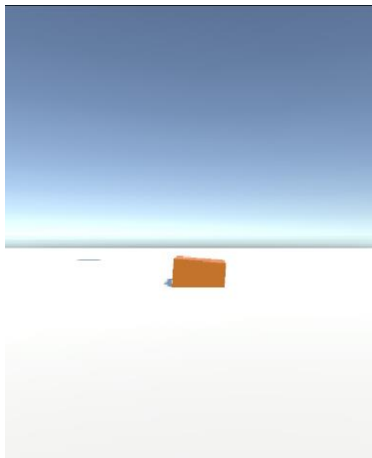
```
void Update()  
{  
    light.intensity = Mathf.Lerp(light.intensity, 8f, 0.5f);  
    light.intensity = Mathf.Lerp(light.intensity, 8f, 0.5f * Time.deltaTime);  
}
```

Destroy

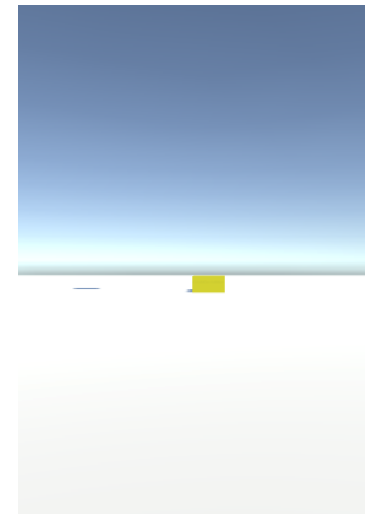
- Use the Destroy() function to remove GameObjects and Components at runtime. A script named DestroyObject attached to yellow object.

```
Destroy(gameObject);
```

```
Destroy(gameObject, 10f);
```



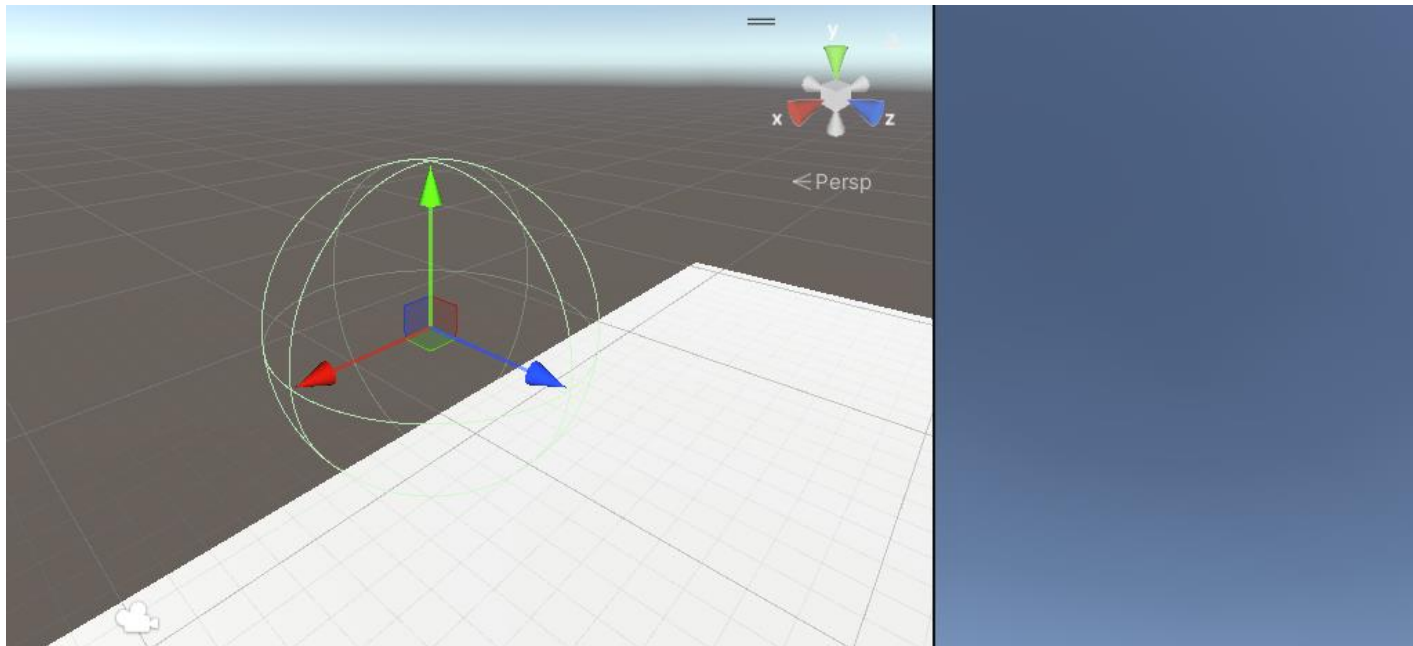
```
public GameObject orange;  
void Start()  
{  
    Destroy(orange, 5f);  
}
```



Destroy Component

- Destroy the mesh render component so that the object is no longer visibly rendered

```
Destroy(GetComponent<MeshRenderer>());
```



Input Manager

The **Input Manager** uses the following types of controls:

- **Key** refers to any key on a physical keyboard, such as W, Shift, or the space bar.
- **Button** refers to any button on a physical controller (for example, gamepads), such as the X button on a remote control.
- A **virtual axis** (plural: **axes**) is mapped to a control, such as a button or a key. When the user activates the control, the axis receives a value in the range of $[-1..1]$. You can use this value in your **scripts**.

Get Key

GetKey/Down/Up

KeyCode.Space



GetButtonDown: **True**

GetButton: **True**

GetButtonUp: **False**

Get Key

GetKey/Down/Up

KeyCode.Space



GetButtonDown: False

GetButton: True

GetButtonUp: False

Get Key

GetKey/Down/Up

KeyCode.Space



GetButtonDown: False

GetButton: False

GetButtonUp: True

Get Key

GetKey/Down/Up

KeyCode.Space



GetButtonDown: False

GetButton: False

GetButtonUp: False

Get Key

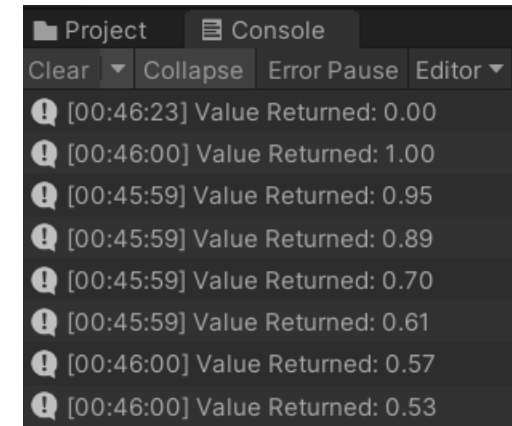
- `Input.GetKeyUp(KeyCode.Space)`
- `Input.GetKeyDown(KeyCode.R)`
- `Input.GetKeyDown(KeyCode.B)`

Get Axis

- Return the float value between minus one and positive one $[-1,1]$
- Access Edit > Project settings > Input > Axes
- Gravity: standard behavior of.GetAxis horizontal, affect how fast the scale return to zero after the button has been released.
- Dead: using a joystick to represent our axis. The higher the dead value is the larger the dead zone.
- Sensitive: control how quickly the return value of the input reach one or minus one. The higher the number the more responsive. The lower the number, the more smooth
- Snap: allow to return zero if both positive and negative are held.

Get Axis

```
public float range = 2;
public Text textOutput ;
void Update()
{
    float h = Input.GetAxis("Horizontal");
    float xPos = h * range;
    transform.position = new Vector3(xPos, 2f, 0);
    Debug.Log("Value Returned: " + h.ToString("F2"));
}
```



Get Axis Exercise

- Vertical axis
- Dual axis

Get Component

- Script as a custom component so GetComponent can access other scripts and components
- `GameObject.GetComponent: T` A reference to a component of the type `T` if one is found, otherwise null.

```
public class UsingComponents : MonoBehaviour
{
    public GameObject otherGameObject;
    private AnotherScript another;
    private AnotherTwoScript anothertwo;
    private BoxCollider boxCol;

    void Awake ()
    {
        another = GetComponent<AnotherScript>();
        anothertwo = otherGameObject.GetComponent<AnotherTwoScript>();
        boxCol = otherGameObject.GetComponent<BoxCollider>();
    }
    void Start()
    {
        boxCol.size = new Vector3(3,3,3);

        Debug.Log("The player's score is " + another.playerScore);
        Debug.Log("The player has died " + anothertwo.numberOfPlayerDeaths + " times");
    }
}
```

Get Component

```
public class AnotherScript : MonoBehaviour
{
    public int playerScore = 9001;
}
```

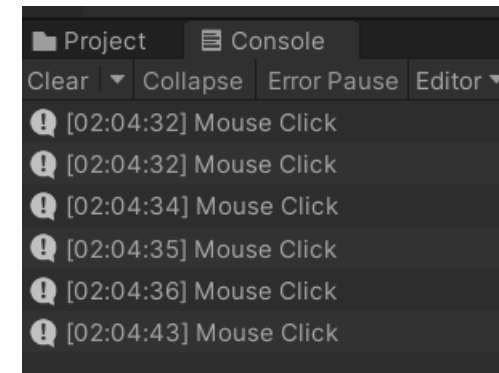
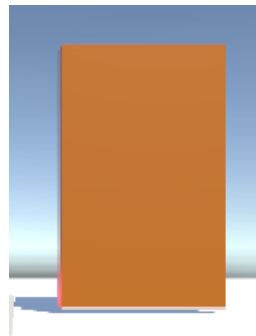
AnotherScript.cs

```
public class AnotherTwoScript : MonoBehaviour
{
    public int numberOfPlayerDeaths = 3;
}
```

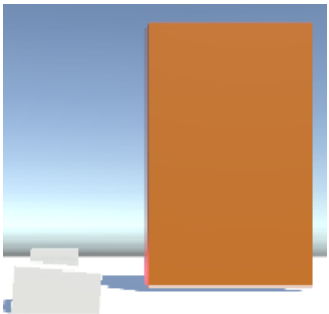
AnotherTwoScript.cs

Mouse Click

```
void OnMouseDown ()  
{  
    Debug.Log("Mouse Click");  
}
```



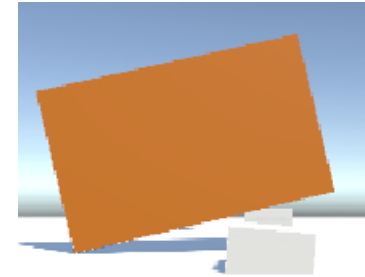
Mouse Click



```
public class MouseClick : MonoBehaviour
{
    private Rigidbody rb;

    private void Awake()
    {
        rb = GetComponent<Rigidbody>();
    }

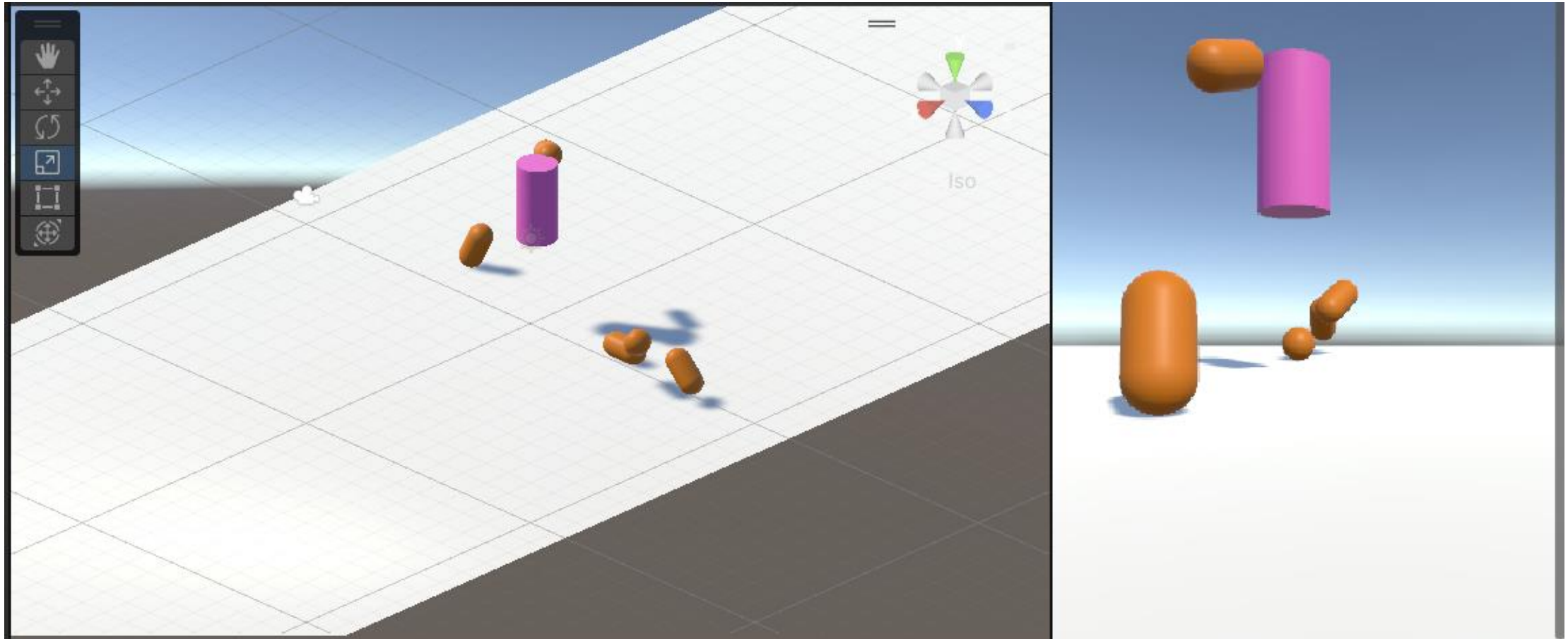
    void OnMouseDown ()
    {
        rb.AddForce(-transform.forward * 200f);
        rb.useGravity = true;
    }
}
```



Instantiate

- A function to create clones of game objects. This is commonly used in the context of cloning a prefab.
- `public static void Destroy(Object obj, float t = 0.0F)`
 - Obj: The object to destroy
 - T: The optional amount of time to delay before destroying the object.

Instantiate



Instantiate

```
public class UsingInstantiate : MonoBehaviour
{
    public Rigidbody projectilePrefab;

    void Update()
    {
        if(Input.GetButtonDown("Fire1"))
        {
            Instantiate(projectilePrefab);
        }
    }
}
```


Instantiate

```
public class UsingInstantiate : MonoBehaviour
{
    public Rigidbody rocketPrefab;
    public Transform barrelEnd;
    void Update()
    {
        if(Input.GetButtonDown("Fire1"))
        {
            Rigidbody rocketInstance;
            rocketInstance = Instantiate(rocketPrefab, barrelEnd.position,
barrelEnd.rotation) as Rigidbody;
            rocketInstance.AddForce(barrelEnd.forward * 5000);
        }
    }
}
```

Destroy Instantiate

```
public class ProjectileDestruction :  
MonoBehaviour  
{  
    // Start is called before the first frame  
    update  
    void Start()  
    {  
        Destroy (gameObject, 1.5f);  
    }  
}
```

Attach script to prelab object

Invoke

- `public void Invoke(string methodName, float time)`
 - `public void InvokeRepeating(string methodName, float time, float repeatRate);`
 - `public void CancelInvoke();`
-
- `methodName`: The name of a method to invoke.
 - `Time`: Start invoking after n seconds.
 - `repeatRate`: Repeat every n seconds.
 - Cancels all `Invoke` calls on this `MonoBehaviour`.

Invoke

```
public class InvokeObject : MonoBehaviour
{
    public GameObject target;
    void Start()
    {
        Invoke ("SpawnObject", 2);
    }

    void SpawnObject()
    {
        Instantiate(target, new Vector3(0, 2, 0),
Quaternion.identity);
    }
}
```

Delay in seconds before
call method: 2

Name of method to call

Exercises

- Call a function named `SpawnObjects` to instantiate randomly object
 - `InvokeRepeating(?,?,?)`
 - Generate `x, y` in range `[1,2]`
 - `Instantiate(target, new Vector3(x, 2, y), Quaternion.identity);`