Contents

Our first script

Unity uses C# as the primary scripting language. You can also write your code in Javascript but most of them prefer C# due to its simplicity. Unity operates only with object-oriented scripting languages. Variables, functions, and classes are the primary building block of any language. The following is a basic script in Unity with a log message.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class HelloWorld : MonoBehaviour
{
    //Variables
    //Functions
    //Classes
    // Start is called before the first frame update
    void Start()
    {
        Debug.Log("Hello World");
    }
    // Update is called once per frame
    void Update()
    {

    }
}
```

1. Variables

It is the name of a memory location which holds values and references of an object. Like any object orient programming language, the variable declaration contains the accessibility level (scope). It can be public, private or protected.

Unity made the object referencing simple by showing the public variables in the inspector. So, the user can easily provide values or references to that variable from the inspector.

If we want to refer to any gameObject in the script then make its visibility public or put a `SerializeField` tag there.

```
public class HelloWorld : MonoBehaviour
{
    public GameObject gameObject1;
    [SerializeField]
    private GameObject gameObject2;
    private GameObject gameObject3;
}
```

2. Classes

Classes are the blueprint of the objects. It is a template that wraps variables and functions together.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;public class HelloWorld : MonoBehaviour
{
    public GameObject gameObject1;
    private GameObject gameObject2;
    // Start is called before the first frame update
    void Start()
    {
        Display();
    }
    // Update is called once per frame
    void Update()
    {

    }
    void Display()
    {
        Debug.Log("Hello World");
    }
}
```

3. Functions

Functions are a set of codes executed to accomplish a specific task. It helps to create reusable codes and implement modularity in your app. Also, we use functions to make our codes readable and portable.

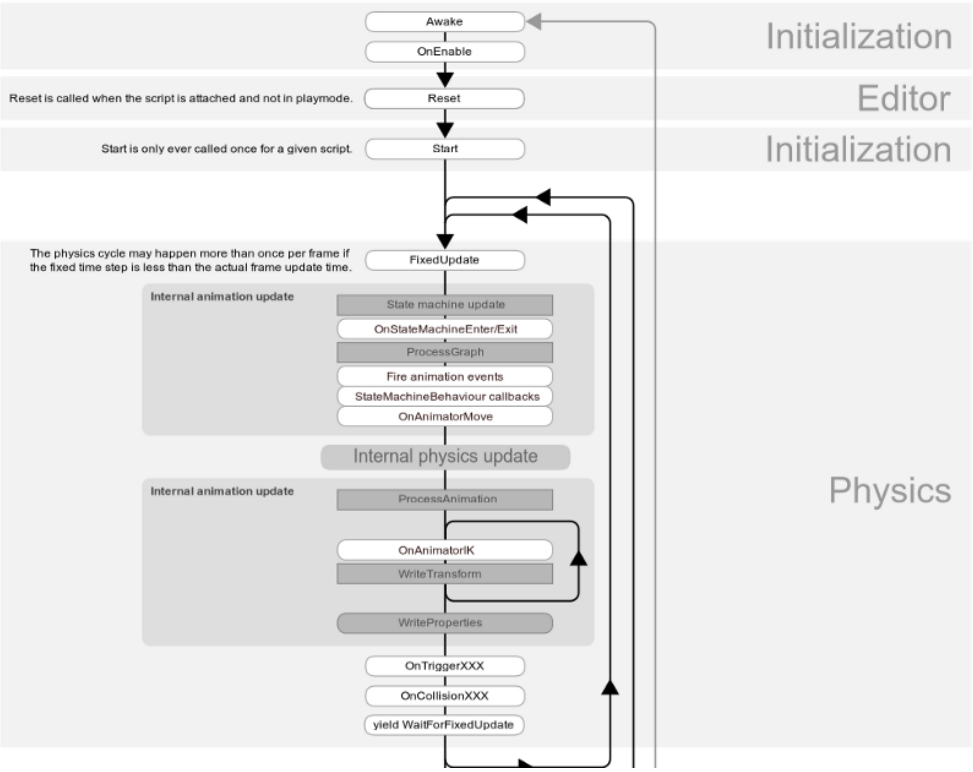
Unity has a couple of life cycle methods such as Awake(), Start(), Update(), and these methods call automatically when some predefined conditions are met.

```
// Called when the script is being loaded
private void Awake() {}
// Called every time the object is enabled
private void OnEnable() {}
// Called on the frame when the script is enabled
private void Start() {}
// Called once per frame
private void Update() {}
// Called every frame after Update
private void LateUpdate() {}
```

```
// Called every Fixed Timestep
private void FixedUpdate() {}
// Called when the renderer is visible by any Camera
private void OnBecameVisible() {}
// Called when the renderer is no longer visible by any Camera
private void OnBecameInvisible() {}
// Allows you to draw Gizmos in the Scene View
private void OnDrawGizmos() {}
// Called multiple times per frame in response to GUI events
private void OnGUI() {}
// Called at the end of a frame when a pause is detected
private void OnApplicationPause() {}
// Called every time the object is disabled
private void OnDisable() {}
// Only called on previously active GameObjects that have been destroyed
private void OnDestroy() {}
```

- **Awake** is called before the scene gets initialized.
- **Start** is called after the scene object creation.
- **Update** is called once per frame.
- **FixedUpdate** is similar to Update but called in equal intervals.
- **LateUpdate** is similar to Update but triggered at end of the frame.

Script lifecycle overview



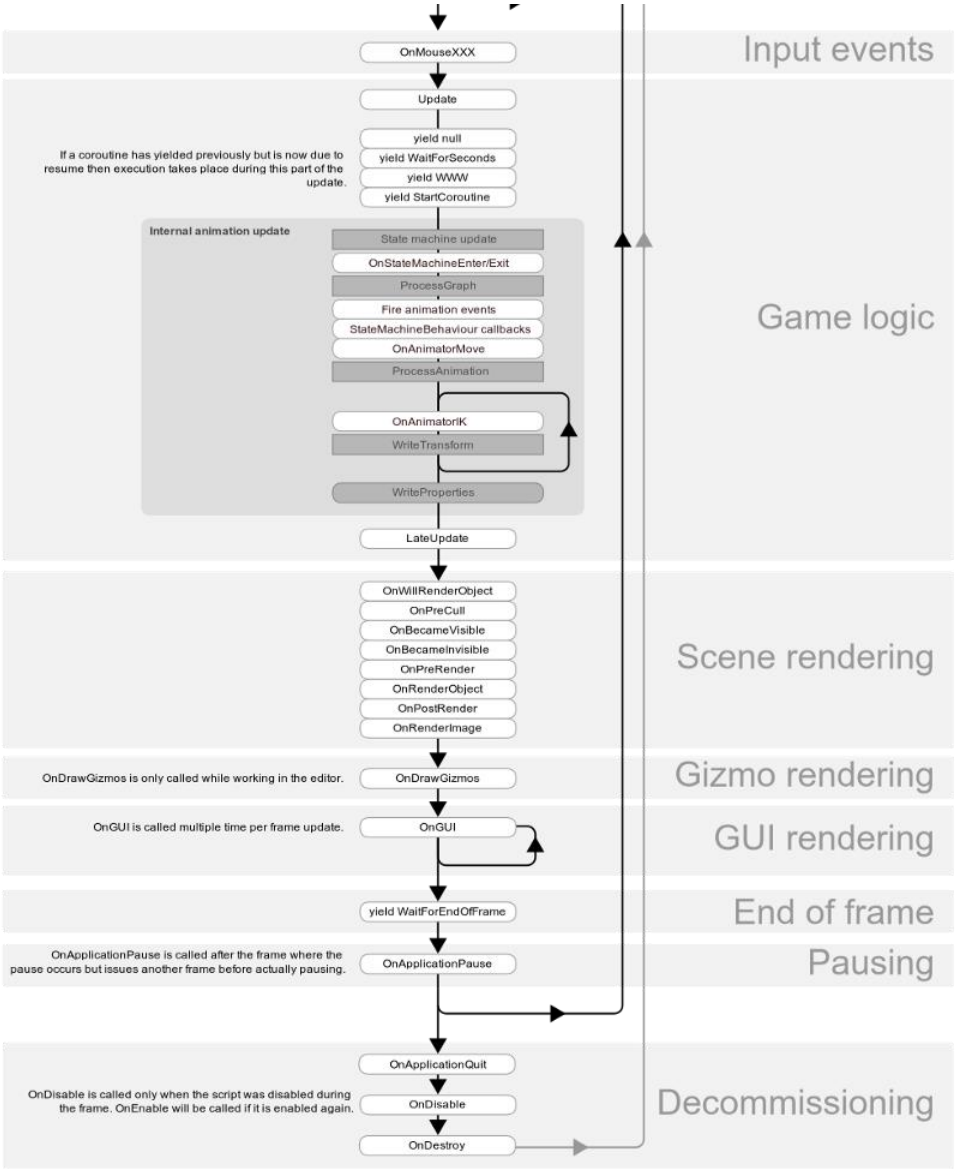


Figure 0-1 Image credits docs.unity3d.com

4. GameObject Manipulation

GameObject is the core component of a Unity project. All objects such as light, UI and 3D models are derived from GameObjects class. This is the parent class for all objects we are using in a unity scene.

In Unity, we can interact with the GameObjects in two ways—through the inspector and using a script. If you want to change the object position then you can easily do it through the inspector window.

```
// Create a GameObject
Instantiate(GameObject prefab);
Instantiate(GameObject prefab, Transform parent);
Instantiate(GameObject prefab, Vector3 position, Quaternion rotation);
Instantiate(bullet);
Instantiate(bullet, bulletSpawn.transform);
Instantiate(bullet, Vector3.zero, Quaternion.identity);
Instantiate(bullet, new Vector3(0, 0, 10), bullet.transform.rotation);

// Destroy a GameObject
Destroy(gameObject);// Finding GameObjects
GameObject myObj = GameObject.Find("NAME IN HIERARCHY");
GameObject myObj = GameObject.FindWithTag("TAG");

// Accessing Components
Example myComponent = GetComponent<Example>();
AudioSource audioSource = GetComponent<AudioSource>();
Rigidbody rgbd = GetComponent<Rigidbody>();
```

5. Input System

The input system is the crucial component in every game that we have played. It might be a keyboard, joystick, or touch. Unity has a Conventional Game Input to access input in your games.

```
if (Input.GetKeyDown(KeyCode.Space)) {
    Debug.Log("Space key was Pressed");
}
if (Input.GetKeyUp(KeyCode.W)) {
    Debug.Log("W key was Released");
}
if (Input.GetKey(KeyCode.UpArrow)) {
    Debug.Log("Up Arrow key is being held down");
}
/* Button Input located under Edit >> Project Settings >> Input */
if (Input.GetButtonDown("ButtonName")) {
    Debug.Log("Button was pressed");
}
if (Input.GetButtonUp("ButtonName")) {
    Debug.Log("Button was released");
}
if (Input.GetButton("ButtonName")) {
    Debug.Log("Button is being held down");
}
```

6. Vector

Vector is a mathematical concept that holds both magnitude and direction. It is useful to describe some properties such as the position and velocity of a moving object in your game, or the distance between two objects. Unity implements Vector2 and Vector3 classes for working with 2D and 3D vectors.

```
Vector3.right /* (1, 0, 0) */   Vector2.right /* (1, 0) */
Vector3.left /* (-1, 0, 0) */   Vector2.left /* (-1, 0) */
Vector3.up /* (0, 1, 0) */      Vector2.up /* (0, 1) */
Vector3.down /* (0, -1, 0) */   Vector2.down /* (0, -1) */
Vector3.forward /* (0, 0, 1) */
Vector3.back /* (0, 0, -1) */
Vector3.zero /* (0, 0, 0) */    Vector2.zero /* (0, 0) */
Vector3.one /* (1, 1, 1) */     Vector2.one /* (1, 1) */
float length = myVector.magnitude /* Length of this Vector */
myVector.normalized /* Keeps direction, but reduces length to 1 */
```

7. Time

Unity supports time-related operations through its Time library. `Time.time`, `Time.deltaTime` and `Time.timeScale` are the most common APIs to work with time in your project.

- `Time.time` returns the time at the beginning of the current frame.
- `Time.deltaTime` returns the time difference between the current and last frame in seconds.
- `Time.timeScale` represents scale at which time elapses.

```
// The time in seconds since the start of the game
float timeSinceStartOfGame = Time.time;
// The scale at which the time is passing

float currentTimeScale = Time.timeScale;
// Pause time
Time.timeScale = 0;
// The time in seconds it took to complete the last frame

// Use with Update() and LateUpdate()
float timePassedSinceLastFrame = Time.deltaTime;
// The interval in seconds at which physics and fixed frame rate
// updates are performed and use with FixedUpdate()

float physicsInterval =  Time.fixedDeltaTime;
```

8. Physics Events

Unity has a sophisticated system to implement physics in your project. It various physics attributes to the `gameObjects` such as gravity, acceleration, collision and other forces.

```
/* Both objects have to have a Collider and one object has to have a
Rigidbody for these Events to work */
private void OnCollisionEnter(Collision hit) {
  Debug.Log(gameObject.name + " hits " + hit.gameObject.name);
}
private void OnCollisionStay(Collision hit) {
  Debug.Log(gameObject.name + " is hitting " + hit.gameObject.name);
}
private void OnCollisionExit(Collision hit) {
  Debug.Log(gameObject.name + " stopped hitting " + hit.gameObject.name);
}

// Trigger must be checked on one of the Colliders
private void OnTriggerEnter(Collider hit) {
  Debug.Log(gameObject.name + " just hit " + hit.name);
}
private void OnTriggerStay(Collider hit) {
  Debug.Log(gameObject.name + " is hitting " + hit.name);
}
private void OnTriggerExit(Collider hit) {
  Debug.Log(gameObject.name + " stopped hitting " + hit.name);
}

// For 2D Colliders
private void OnCollisionEnter2D(Collision2D hit) { }
private void OnCollisionStay2D(Collision2D hit) { }
private void OnCollisionExit2D(Collision2D hit) { }
private void OnTriggerEnter2D(Collider2D hit) { }
private void OnTriggerStay2D(Collider2D hit) { }
private void OnTriggerExit2D(Collider2D hit) { }

// Ray casting to detect the collision
Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
RaycastHit hit;
if (Physics.Raycast(ray, out hit, 100)){
  Debug.DrawLine(ray.origin, hit.point);
  Debug.Log("Hit: " + hit.collider.name);
}
```

9. Rendering materials

Materials tell how a surface should be rendered in the scene. The material contains references to shaders, textures, color, emission and more. Every material requires a shader for rendering the content and the attributes available for that may vary on different shader values.

```
  [SerializeField] Material material;
  [SerializeField] Texture2D texture;
  [SerializeField] Color color = Color.red;

  // Start is called before the first frame update
  void Start() {
      MeshRenderer meshRenderer = GetComponent<MeshRenderer>();

      // Changing material, texture, color and shader at runtime
      meshRenderer.material = material;
      meshRenderer.material.mainTexture = texture;
      meshRenderer.material.color = color;
      meshRenderer.material.SetColor("_Color", Color.blue);
      meshRenderer.material.EnableKeyword("_EMISSION");
      meshRenderer.material.SetColor("_EmissionColor", Color.yellow);
      meshRenderer.material.shader = Shader.Find("Standard (Specular
setup)");
  }
```

10. Lighting

Lighting is a mandatory component in any Unity scene. All unity scenes contain a directional light component in it by default. Unity has four types of lights—directional, points, spot and area lights

```
  [SerializeField] LightType lightType = LightType.Directional;
  Light lightComp = null;

  // Start is called before the first frame update
  void Start() {
     GameObject lightGameObject = new GameObject("The Light");
     lightComp = lightGameObject.AddComponent<Light>();
     lightComp.color = Color.blue;
     lightComp.type = lightType;
     lightGameObject.transform.position = new Vector3(0, 5, 0);
  }

  void Update() {
     if (Input.GetKey(KeyCode.UpArrow))
        lightComp.GetComponent<Light>().enabled = true;
     if (Input.GetKey(KeyCode.DownArrow))
        lightComp.GetComponent<Light>().enabled = false;
  }
```

11. Coroutine

A coroutine is like a background activity which can hold the execution of codes after the yield statement until it returns a value.

```
// Create a Coroutine
private IEnumerator CountSeconds(int count = 10)
{
  for (int i = 0; i <= count; i++) {
    Debug.Log(i + " second(s) have passed");
    yield return new WaitForSeconds(1.0f);
  }
}

// Call a Coroutine
StartCoroutine(CountSeconds());
StartCoroutine(CountSeconds(10));

// Store and call a Coroutine from a variable
private IEnumerator countSecondsCoroutine;
countSecondsCoroutine = CountSeconds();
StartCoroutine(countSecondsCoroutine);

// Stop a stored Coroutine
StopCoroutine(countSecondsCoroutine);
// Coroutine Return Types
// Waits until the next Update() call
yield return null;
// Waits until the next FixedUpdate() call
yield return new WaitForFixedUpdate();
// Waits until everything this frame has executed
yield return new WaitForEndOfFrame();
// Waits for game time in seconds
yield return new WaitForSeconds(float seconds);
// Waits until a custom condition is met
yield return new WaitUntil(() => MY_CONDITION);
// Waits for a web request
yield return new WWW("MY/WEB/REQUEST");
// Waits until another Coroutine is completed
yield return StartCoroutine("MY_COROUTINE");
```

12. Animation

It is an easy task to create animations in Unity. Unity made it simple with the help of Animator controls and Animation Graph. Unity calls the animator controllers to handle which animations to play and when to play them. The animation component is used to playback animations.

It is quite simple to interact with the animation from a script. First, you have to refer the animation clips to the animation component. Then get the `Animator` component reference in the script via `GetComponent` method or by making that variable public. Finally, set `enabled` attribute value to `true` for enabling the animation and `false` for disabling it.

```
[SerializeField] GameObject cube;
cube.GetComponent<Animator>().enabled = true;
cube.GetComponent<Animator>().SetTrigger("Enable");
cube.GetComponent<Animator>().ResetTrigger("Enable");
cube.GetComponent<Animator>().SetInteger("animId", 1);
```

13. Hotkeys

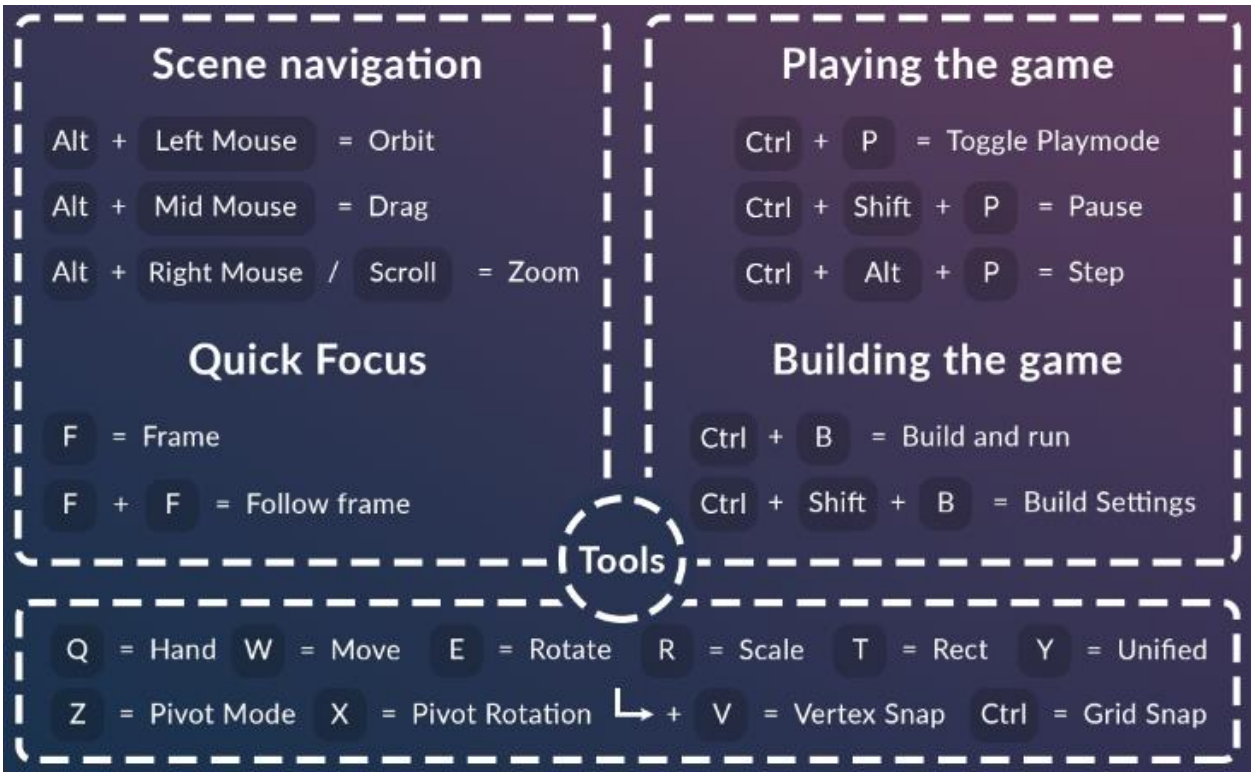


Image credits patreon.com