Unity

An Introduction to the Unity Engine

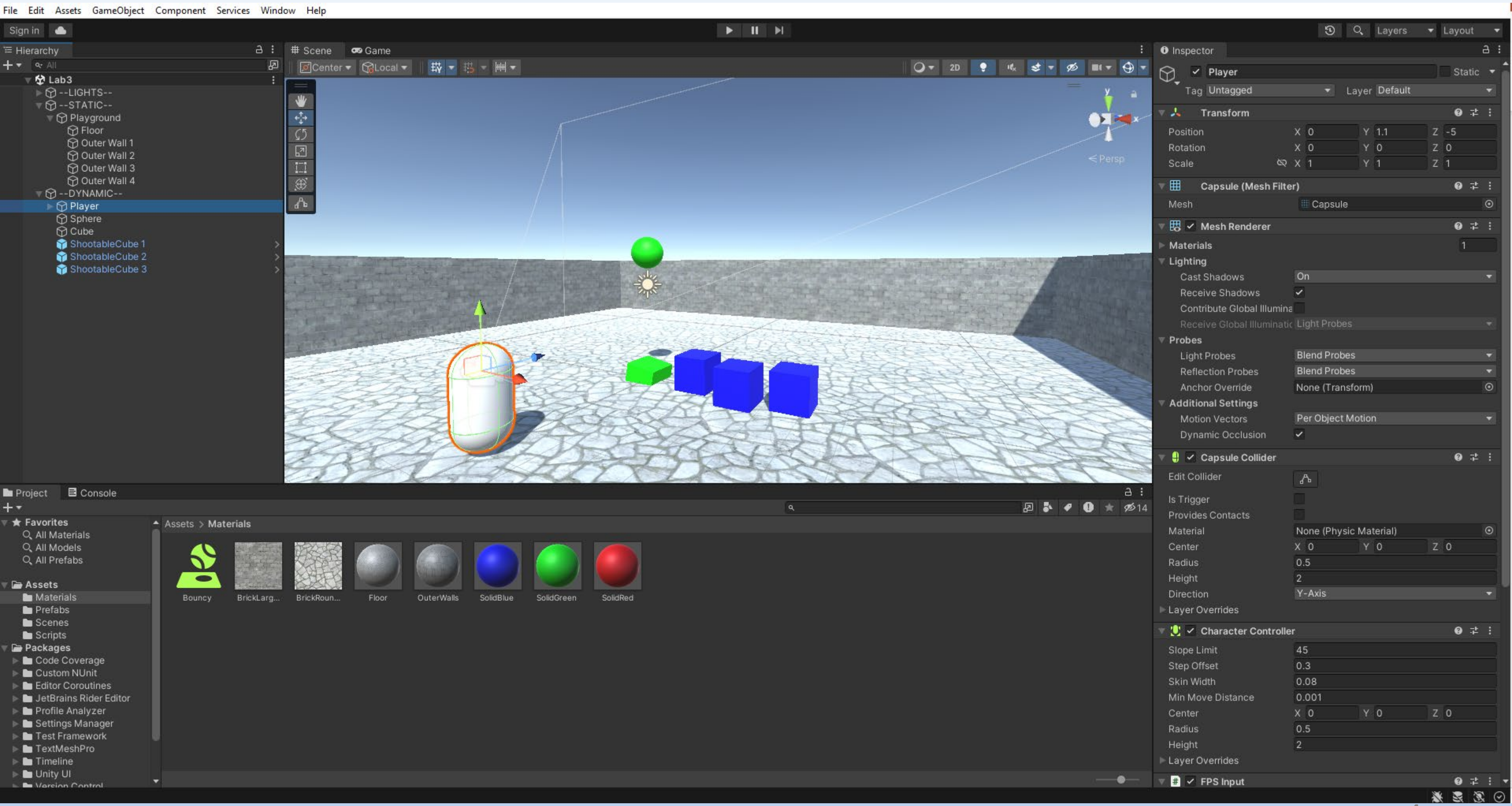
Overview

- The Unity Interface
- Essential concepts
  - ➢ GameObject – Component
  - ➢ Material
- Scripting concepts
  - ➢ Some important classes
  - ➢ Event processing
  - ➢ Input

Unity

- Unity engine
  - ➢ Real-time 3D development engine
    - Windows, Mac, and Linux
    - Desktop, consoles, mobile, AR/VR
  - ➢ Used in industry for various applications
    - 2D/3D games, film and cinematics, architecture and engineering, automotive and transportation, data visualisation, etc.
  - ➢ Popular general-purpose engine
    - Especially for mobile games
    - Recommended by Microsoft for its HoloLens platform
    - Example games
      - – Pokemon Go, Call of Duty: Mobile, Beat Saber, Hearthstone

Unity Editor Interface



Hierarchy

- Lab3
  - --LIGHTS--
  - --STATIC--
    - Playground
      - Floor
      - Outer Wall 1
      - Outer Wall 2
      - Outer Wall 3
      - Outer Wall 4
  - --DYNAMIC--
    - Player
    - Sphere
    - Cube
    - ShootableCube 1
    - ShootableCube 2
    - ShootableCube 3

Project

- Favorites
  - All Materials
  - All Models
  - All Prefabs
- Assets
  - Materials
  - Prefabs
  - Scenes
  - Scripts
- Packages
  - Code Coverage
  - Custom NUnit
  - Editor Coroutines
  - JetBrains Rider Editor
  - Profile Analyzer
  - Settings Manager
  - Test Framework
  - TextMeshPro
  - Timeline
  - Unity UI
  - Version Control

Assets > Materials

Bouncy, BrickLarg..., BrickRoun..., Floor, OuterWalls, SolidBlue, SolidGreen, SolidRed

Inspector

Player — Static

Tag Untagged — Layer Default

Transform

Position	X 0	Y 1.1	Z -5
Rotation	X 0	Y 0	Z 0
Scale	X 1	Y 1	Z 1

Capsule (Mesh Filter)

Mesh	Capsule

Mesh Renderer

Materials	1
Lighting	
Cast Shadows	On
Receive Shadows	✓
Contribute Global Illumina	
Receive Global Illuminatic	Light Probes
Probes	
Light Probes	Blend Probes
Reflection Probes	Blend Probes
Anchor Override	None (Transform)
Additional Settings	
Motion Vectors	Per Object Motion
Dynamic Occlusion	✓

Capsule Collider

Edit Collider	
Is Trigger	
Provides Contacts	
Material	None (Physic Material)
Center	X 0 Y 0 Z 0
Radius	0.5
Height	2
Direction	Y-Axis
Layer Overrides	

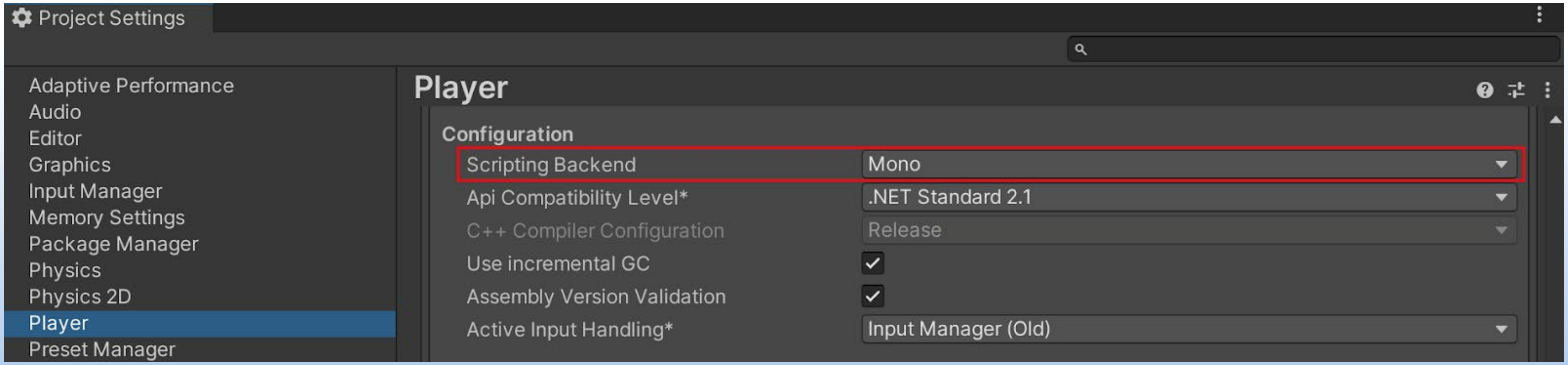
Character Controller

Slope Limit	45
Step Offset	0.3
Skin Width	0.08
Min Move Distance	0.001
Center	X 0 Y 0 Z 0
Radius	0.5
Height	2
Layer Overrides	

FPS Input

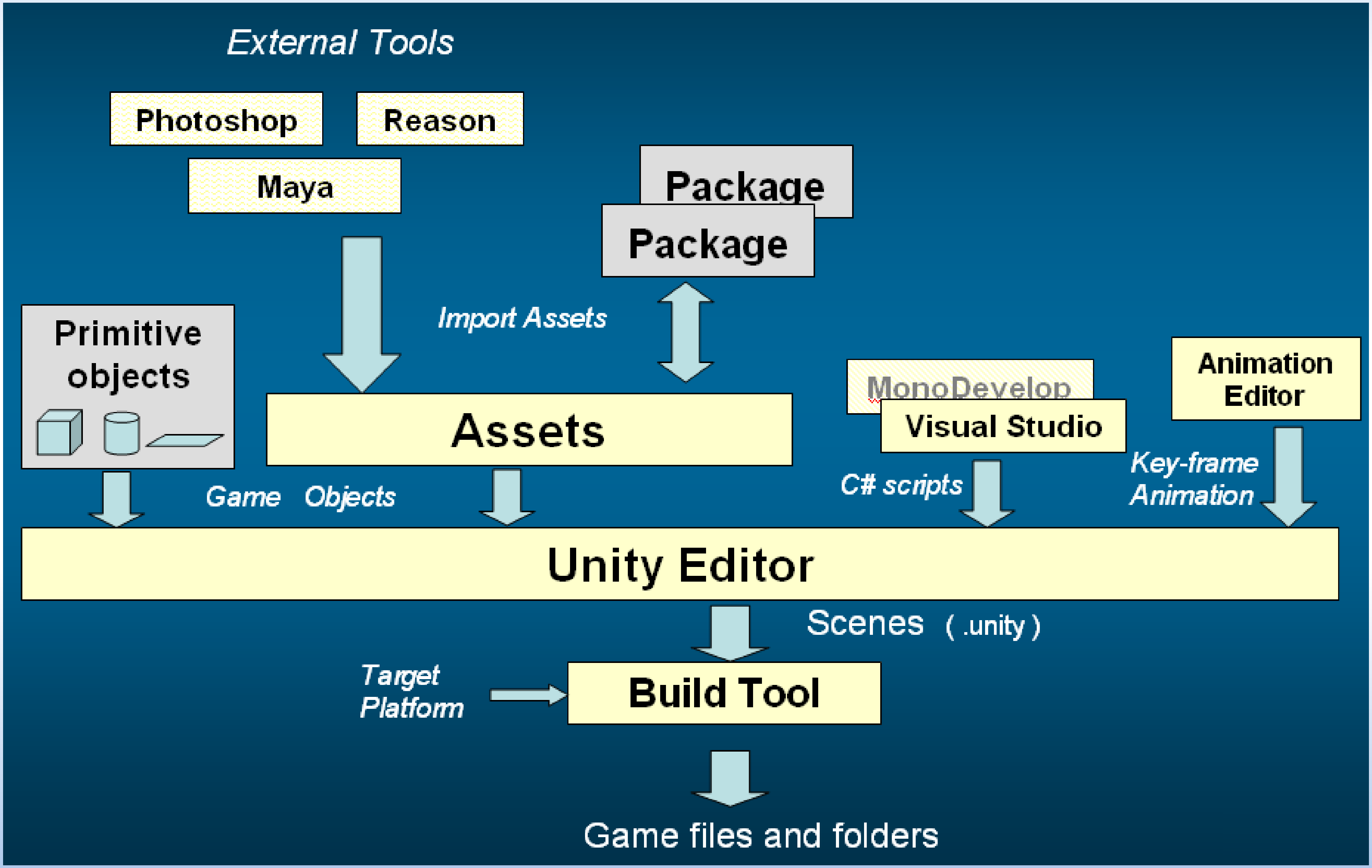
Multi-Platform Support

- Unity uses the open-source .NET platform
  - ➢ Ensures applications made with Unity can run on a wide variety of different hardware configurations
  - ➢ .NET platform supports a range of languages and API libraries
  - ➢ Unity uses the Mono backend by default
  - ➢ Supports C# language natively



Scripting Backend	Mono
Api Compatibility Level*	.NET Standard 2.1
C++ Compiler Configuration	Release
Use incremental GC	✓
Assembly Version Validation	✓
Active Input Handling*	Input Manager (Old)

Unity Development Pipeline

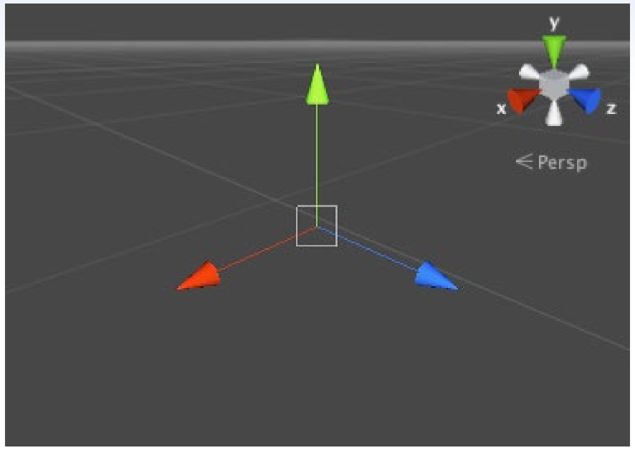


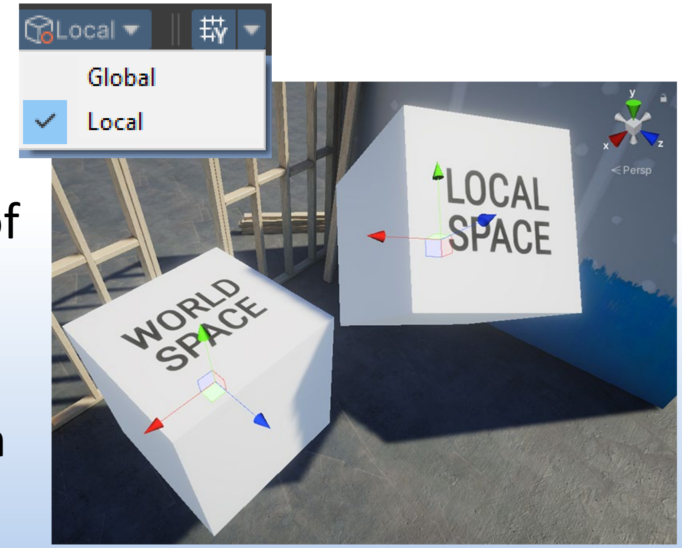
Unity Essentials

- **Scenes**
  - Scenes are where you work with content in Unity
  - Assets that contain all or part of a game or application
    - A simple game might only require a single scene
    - For more complex games, might use one scene per level
      - Each with its own environments, characters, obstacles, decorations, and UI
  - Can create any number of scenes in a project
  - Unity’s default scale
    - 1 unit = 1 metre

Unity Essentials

- ➢ Left-handed coordinate system
  - Y-axis is up
- ➢ World/global space
  - Coordinate system of the scene itself
  - Origin is the centre of the scene
  - Cannot change the direction of this coordinate system
- ➢ Local space
  - Coordinate system relative to rotation of a specific object
  - Origin is at the object's pivot point
  - Its axes will change depending on which direction it is facing





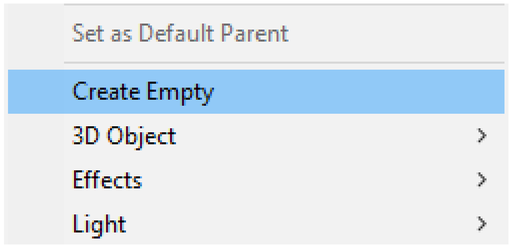
https://www.techarthub.com/a-guide-to-unitys-coordinate-system-with-practical-examples/

Unity Essentials

- **GameObject**
  - Unity’s GameObject class represents anything that can exist in a Scene
    - The base class of all entities in Unity scenes
    - Building blocks for scenes
  - Acts as a container for **Components**
    - Determine how the GameObject looks and what it does
    - Always has a Transform component
      - Position, rotation and scale
  - Provides a collection of methods
    - Can work with these in code, e.g.,
      - Setting and checking properties
      - Adding/removing components

Unity Essentials

- Can have parent-child relationships
  - Good practice to reset a GameObject’s transform before assigning parent-child
- Empty GameObject
  - Not visible in the scene
  - Some uses
    - Organise objects in the scene by separating GameObjects into different categories
    - Attach scripts that are not directly related to objects in the scene
    - Parent transform, e.g., pivot point



Unity Essentials

- **Components**
  - ➢ Functional pieces of every GameObject
    - Contain editable properties that define the behaviour of a GameObject
    - With a GameObject selected, components attached to it and their properties appear in the Inspector window
  - ➢ Can attach many components to a GameObject
    - Every GameObject can only have one Transform component
    - Composition relationship rather than inheritance
  - ➢ Create a component with a script to customise behaviour

Unity Essentials

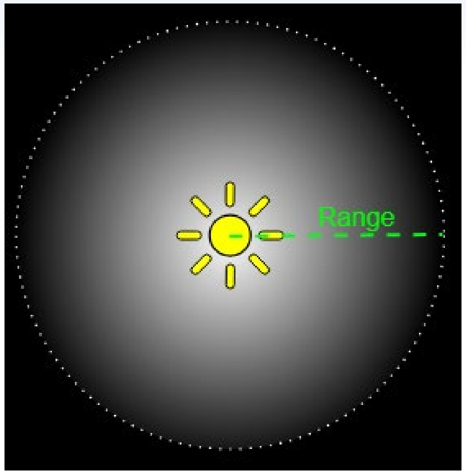
- **Material**
  - ➢ An asset that controls the appearance of a surface
  - ➢ Contains a reference to a Shader object
    - A shader program that runs on the GPU
    - Most common type of shader forms part of the graphics pipeline
  - ➢ Can assign colour and textures using the Albedo property
- Textures
  - ➢ A texture is a bitmap image applied to a surface
  - ➢ Applied to objects using Material
  - ➢ Texture dimensions should be to the power of two
    - E.g., 32x32, 64x64, 128x128, 256x256, etc.

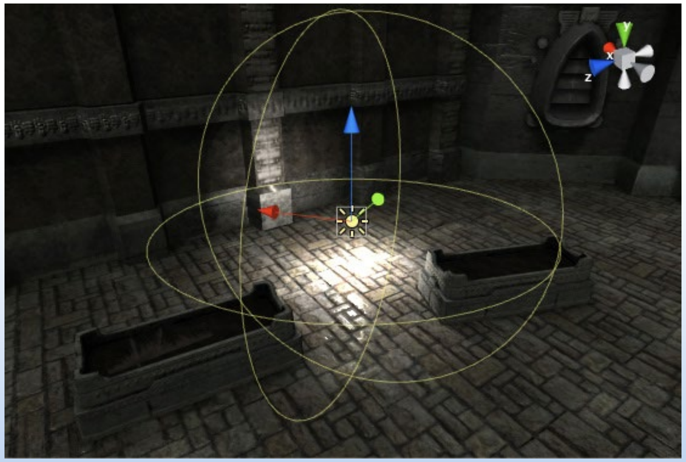
Unity Essentials

- **Prefab**
  - ➢ To reuse a GameObject several times
    - Duplicate by copying – all editable independently
  - ➢ Prefab asset type
    - Any edits made to a prefab are reflected in all instances produced from it
  - ➢ Acts as a template for a GameObject
    - Includes its components
  - ➢ Appears in blue the Hierarchy window
  - ➢ A copy of a prefab is known as an *instance*
  - ➢ Can instantiate prefab in scripts

Unity Essentials

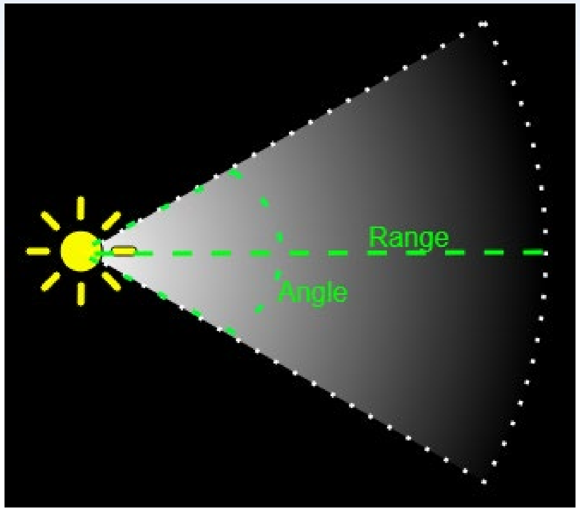
- **Light**
  - ➢ Type property
    - Point light
      - – Located at a point in the Scene and emits light in all directions equally
      - – The intensity diminishes with distance from the light, reaching zero at a specified range
      - – Useful for simulating lamps and other local sources of light in a scene
      - – Can also use them to make a spark or explosion illuminate its surroundings

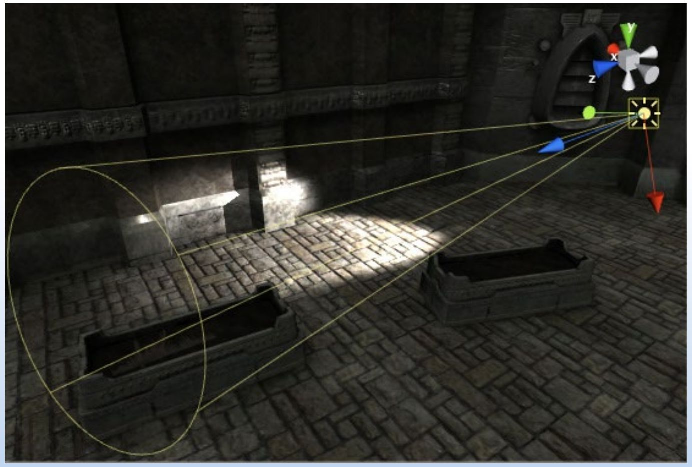


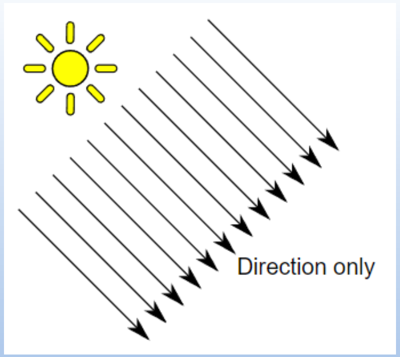


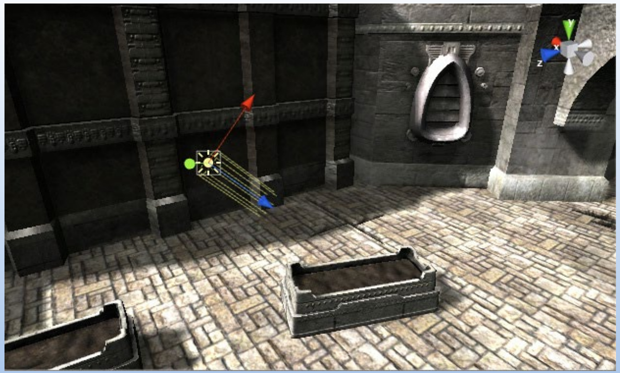
Unity Essentials

- Spot light
  - A specified location and range over which the light falls off
  - Constrained to an angle, resulting in a cone-shaped region of illumination
  - Used for artificial light sources such as flashlights, car headlights and searchlights
- Directional light
  - Located infinitely far away and emits light in one direction only









Scripts

- Scripting
  - ➢ Create your own components using scripts
    - Trigger game events, modify component properties over time, respond to user input, etc.
    - On creation, the name of the new script used as the class name
      - – Must be the same to enable the script component to be attached
    - Scripts are a kind of blueprint
      - – When attached to a GameObject, it creates a new instance of the object defined by the blueprint
  - ➢ Initialisation is not done using a constructor
    - Construction of objects handled by the editor
    - **Do not define a constructor**
      - – Defining a constructor for a script component will interfere with the normal Unity operation

Scripts

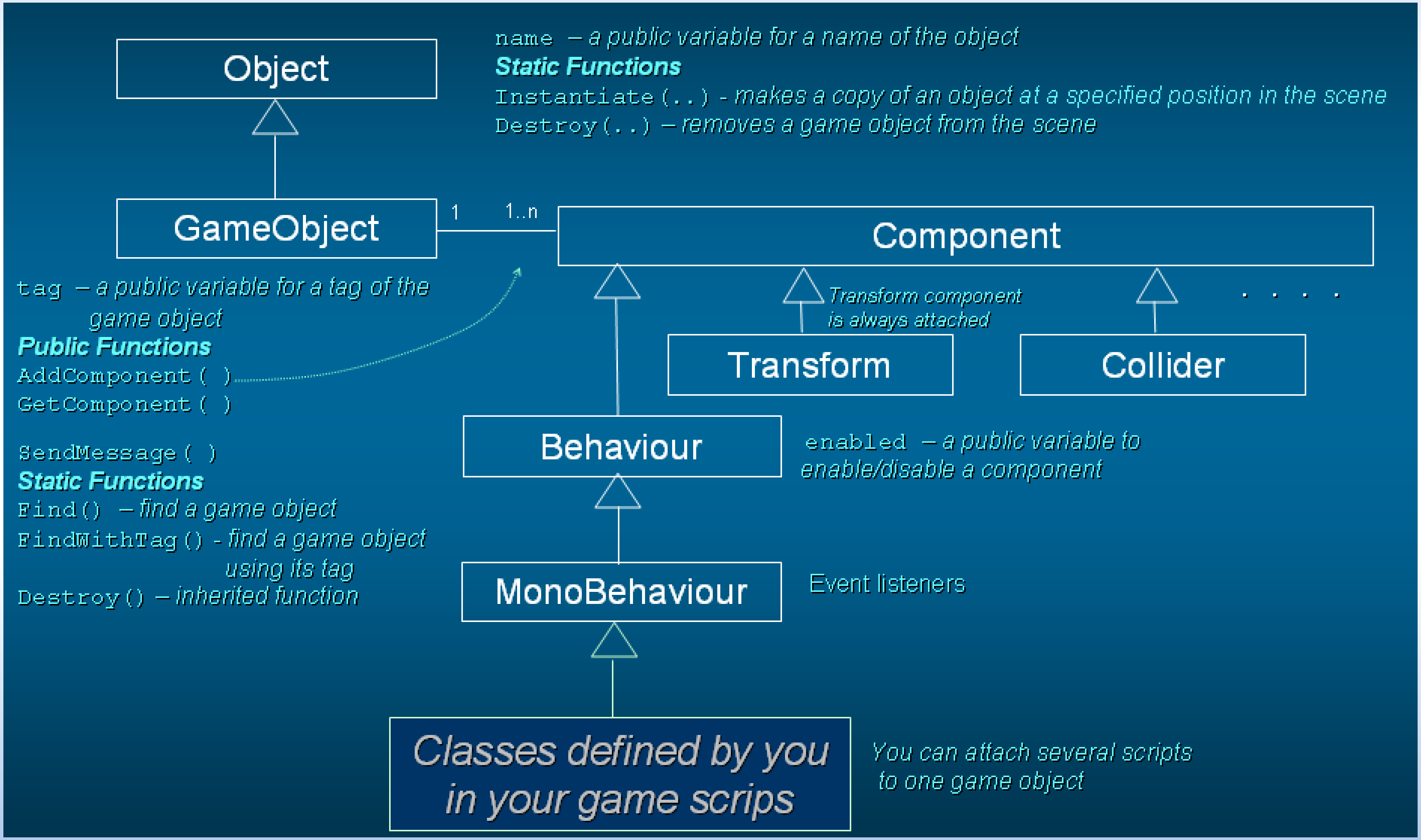
- Anatomy of a script file
  - ➢ Derived from MonoBehaviour

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

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

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

Hierarchy of Classes


Object
GameObject
Component
Transform
Collider
Behaviour
MonoBehaviour
Classes defined by you in your game scrips
1
1..n
. . . .
Transform component is always attached
Event listeners
You can attach several scripts to one game object

`name` – *a public variable for a name of the object*

Static Functions

`Instantiate(..)` - *makes a copy of an object at a specified position in the scene*

`Destroy(..)` – *removes a game object from the scene*

`tag` – *a public variable for a tag of the game object*

Public Functions

`AddComponent( )`

`GetComponent( )`

`SendMessage( )`

Static Functions

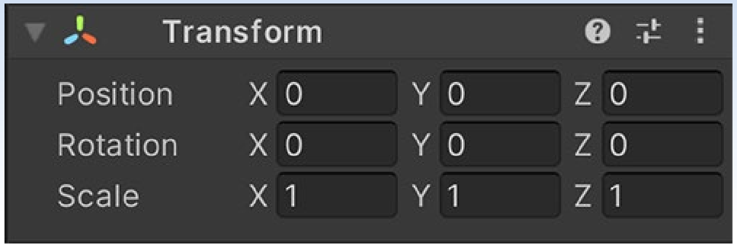
`Find()` – *find a game object*

`FindWithTag()` - *find a game object using its tag*

`Destroy()` – *inherited function*

`enabled` – *a public variable to enable/disable a component*

Scripts


Transform

Position	X 0	Y 0	Z 0
Rotation	X 0	Y 0	Z 0
Scale	X 1	Y 1	Z 1

- Transform class
  - ➢ Store and manipulate position, rotation and scale
  - ➢ Some public variables
    - `Position` – a Vector3 that stores position in world space
    - `Rotation` – a Quaternion that stores rotation in world space
    - `gameObject` – the GameObject this component is attached to
  - ➢ Some public methods
    - `Translate()` – move the transform by the translation direction and distance
    - `Rotate()` – applies a rotation, often in Euler angles
    - `LookAt()` – rotates the transform so the forward vector points at target's current position

Scripts

- MonoBehaviour class
  - ➢ The base class that many Unity scripts are derived from
  - ➢ Offers life cycle functions for easier development
  - ➢ Always exist as a component of a GameObject and can be instantiated with `GameObject.AddComponent()`
  - ➢ Can be deleted with `Object.Destroy()`
    - The C# object remains in memory until garbage is collected
    - A MonoBehaviour in this state acts as if it is null
  - ➢ Some inherited members
    - `transform` – transform component attached to this object
    - `gameObject` – the GameObject this component is attached to

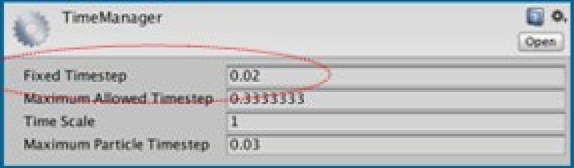
Scripts

- Some public methods
  - `GetComponent()` – gets a reference to a component of type T on the same GameObject
  - `SendMessage()` – calls a named method on every MonoBehaviour in this GameObject
- Some event functions
  - `Start()` – called on the frame when a script is enabled
  - `Update()` – called every frame, if the MonoBehaviour is enabled
  - `OnMouseDown()` – called when the user presses a mouse button over object's collider
  - `OnCollisionEnter()` – called when this collider touches another collider

Event processing loop

Controlled by the Event/Messaging System of the Gameplay Foundations module

The physics loop is synchronised by a timer that is independent to the frame rate

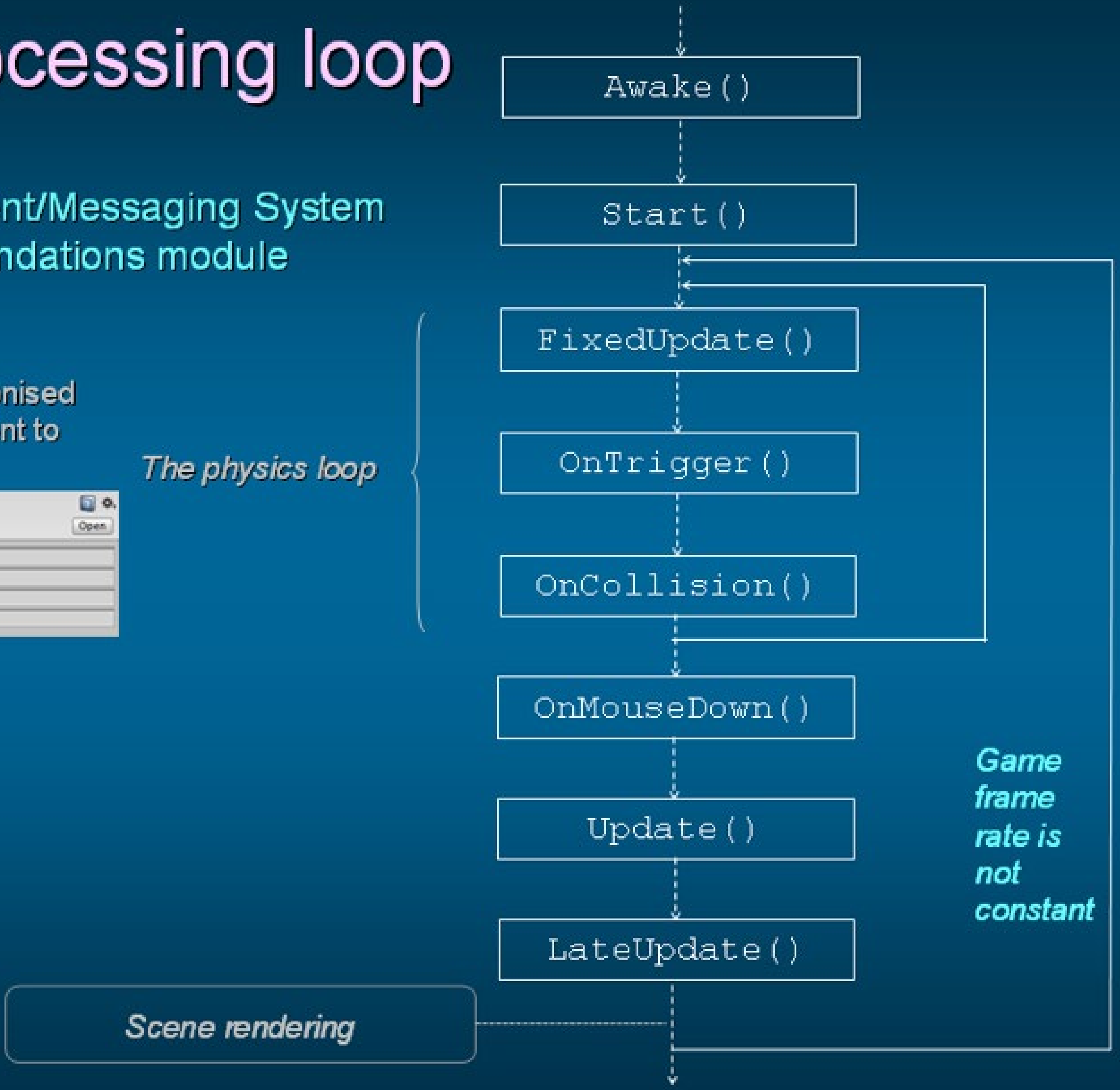


TimeManager

Open

Fixed Timestep	0.02
Maximum Allowed Timestep	0.3333333
Time Scale	1
Maximum Particle Timestep	0.03

Edit -> Project Settings -> Time



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

Scripts

- Time class
  - ➢ Provides numeric values to measure time elapsing while game is running
  - ➢ Some important properties
    - `Time.time` – read-only, time (in seconds) since project started playing
    - `Time.deltaTime` – read-only, time (in seconds) elapsed since the last frame. Varies depending on the frames per second rate
    - `Time.timeScale` – controls the rate at which time elapses
    - `Time.fixedDeltaTime` – controls the interval of Unity's fixed timestep loop (for physics)

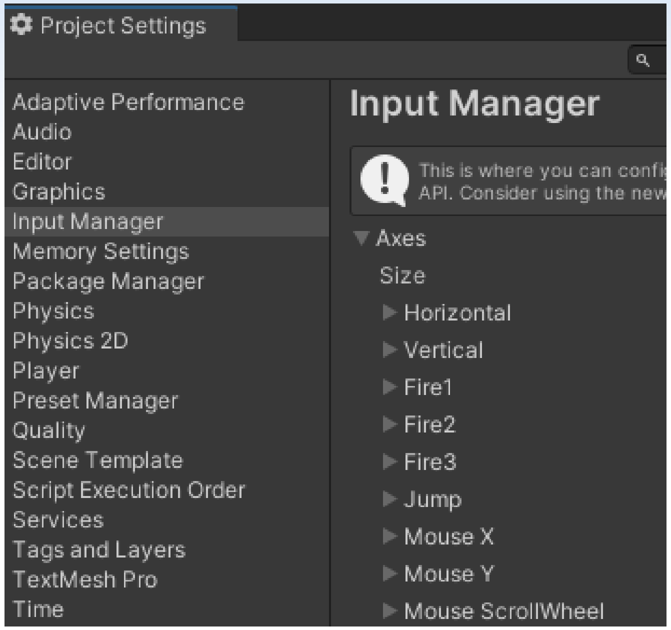
Scripts

- Input
  - ➢ Every project has several input axes created by default
    - Enables keyboard, mouse, and joystick input regardless of device
  - ➢ An axis receives a value in the range of [–1..1]

```
float move = Input.GetAxis(“Horizontal”);
if(Input.GetButton("Fire1"))
```

  - ➢ Alternatively, can get input directly

```
float left = Input.GetKey(KeyCode.A);
if(Input.GetMouseButton(0))
```



This is where you can confi
API. Consider using the new

Scripts

- Polling – regularly check current input state

```
void Update() {
    // check for left mouse click every frame
    if( Input.GetMouseButton(0) ) {
        // ...process input ...
    }
}
```

- Event processing – associate input with an event. Program execution is interrupted on input to process the event

```
// mouse button pressed when cursor is over Collider
void OnMouseDown() {
    // ...process input ...
}
```

Scripts

- Difference between
  - `GetKey()` – returns true while key is held down
  - `GetKeyDown()` – returns true the *first frame* key pressed
  - `GetKeyUp()` – returns true the *first frame* key released
- Similarly,
  - `GetMouseButton()`, `GetMouseButtonDown()`, `GetMouseButtonUp()`

Scripts

- Serialising
  - ➢ Public variables are displayed in the Inspector
    - Can change their values at runtime
  - ➢ Instead of using public variables
    - Using `[SerializeField]` makes the variable appear in the Inspector, but is a private variable
- Component dependencies
  - ➢ A component might depend on other components being attached to the GameObject
  - ➢ Can enforce dependency, e.g., `[RequireComponent(typeof(Rigidbody))]`

Scripts

- **Coroutines**
  - ➢ When a normal method is called
    - It runs to completion, then returns control to the calling method
    - Any action that takes place within the method must happen within a single frame update
  - ➢ A coroutine
    - A method that can pause its execution, return control to Unity, then continues where it left off on the following frame
    - Allows a task to be spread across several frames

Scripts

- Declare coroutine

```
IEnumerator Fade() {
    Color c = renderer.material.color;
    for (float alpha = 1f; alpha >= 0; alpha -= 0.1f) {
        c.a = alpha;
        renderer.material.color = c;
        yield return new WaitForSeconds(.1f);
    }
 }
```

- Run coroutine

```
void Update() {
    if (Input.GetKeyDown("f")) {
        StartCoroutine(Fade());
    }
}
```

Scripts

- Interaction between objects
  - Messaging system
    - Call a method that is implemented in another script attached to the same object, or another object, and pass a parameter to it
  - Public method
    - Get a component of the target object (which is a script) using the component’s name and call its public method explicitly

Scripts

```
// A script attached to "enemy" object
. . .
public int health = 5;
   .    .     .
void HitByLaser ( int damage ) {
        health -= damage;
}
```

```
  .    .    .
private RaycastHit hitInfo;  // a structure initialized by Raycast() if the ray hits an object
  .    .    .
void Update () {
    .    .    .
     if ( Physics.Raycast (transform.position, directiononOfFire, out hitInfo, 20) )  {
            hitInfo.transform.SendMessage( "HitByLaser",  hDamage );
     }
    .    .    .
    // another way to call HitByBullet() that is a member function of another class
    GameObject.Find("enemy").SendMessage("HitByLaser", hDamage);
}
```

Scripts

```
public class TargetHit : MonoBehaviour {
   public int health = 5;
   .    .    .
    public void HitByBullet ( int damage ) {
        health -= damage;
    }
}
```

```
 .      .      .
private RaycastHit hitInfo;
  .    .        .
void Update () {
    .    .      .
      if( Physics.Raycast (transform.position, directiononOfFire, out hitInfo, 20) )  {
            TargetHit  target = hitInfo.transform.gameObject.GetComponent<TargetHit>();
            target.HitByBullet( hDamage );
      }
    .     .     .
}
```

References

- Among others, material sourced from
  - https://unity.com/
  - https://docs.unity3d.com
  - Jason Gregory, Game Engine Architecture, A.K. Peters
  - Will Goldstone, Unity Game Development Essentials