# Unity

## What is Unity?

* Laying on top of MonoDevelop platform
  + MonoDevelop is .Net implementation that allows to build for various platforms
  + MonoDevelop + whole bunch of game related libraries + Editor UI = Unity Game Engine
  + Treat MonoDevelop as a black box, no need of detailed knowledge
  + Laying on top of also IL2CPP scripting backend.
    - Developed by Unity
    - Created to support the latest platforms and OS versions (for ex iOS 11+ & latest Android)
    - Offers improvements

## Supported Languages

* + C#
    - Most used and recommended.
    - Most of the .Net libraries has their equivalents, but some functionalities varies depending on the current chosen platform for ex. Android, iOS, Windows etc..
    - Some of the .Net libraries has their "Unity" versions that are a bit tweaked, with better suited for gaming functions. For ex. Math(.Net) - Mathf (Unity).
  + JS - Not really JavaScript, more like a Unity scripting language. Not covered by the course. Not supported by Unity anymore.
  + Boo - Not covered by the course. Not supported by Unity anymore.

## Positioning and Coordinate System

* Cartesian Coordinate System
  + Most recognized and used. X/Y/Z. Remember Y is upwards, Z is forward.
  + Vector3 coodinates used. Remember when moving an object only horizontally we use x/z axes.
* Local/World coordinates
  + Let's say we hold a box in front of us. Now we rotate the box on the Y axis with 90 degrees. Now our body is still pointing forward but the box is pointing on the right.
    - Its own forward direction is not anymore the same as the global/world forward (ours).
    - If we write code to move the children forward it will move to the right, because it is pointing there now.
    - When we rotate our body we also rotate the box (parent-child relation)
* Rotation
  + Euler - Vector3 - simply degrees for each axis
  + Quaternion - it has few super long explanations for the mathematical representation of it. It is basically Vector4. Refrain from directly assiging the components of it. There is more than enough helping functionallities in the libraries to convert Quaternion to Vector3 (Euler) and vice versa. All needed calculations can and will be done using Vector3 and then converted to quaternion. Unity internally uses quaternions, but we don't need to work with them.

## Cameras

* + Defines what we see in the final game
  + We can have as many as we want
  + We can have one camera to render only certain parts and another for other things
  + Most important properties
    - Backgorund - Simply what are we going to see when there is nothing in front of the camera
    - Culling mask - filter what will that paticular camera render
    - Field Of View - The angle of perspective
    - Clipping planes - how far and close will the camera "see"

## Lights and Shadows

* Light Types
  + Point - point and radius in which it works
  + Spot - like a nightstand lamp
  + Directional - simulates Sun with direction
  + Area - like point, but in rectangle from one direction to another
  + Ambient - environment light - no specific source

## Shaders and Materials

* + Shaders - instructions/script for the GPU on how to calculate the color for each pixel being rendered
  + Textures - bitmap images that needs to be showed
  + Materials - the glue between the Shader and the Texture

## GameObjects and Components

* + GameObject is a container for components
  + GameObject is a class inheriting Object
  + Component is a class that is intended to be attached to a GameObject
  + When we attach component to a GameObject upon scene start it is like we say - Component c = new Component();
  + When we make script to manupulate a GameObject we make that script to inherit MonoBehavior in order to be able use Unity functionalities and to be able to attach it to the GameObject

## Scene, Hierarchy and Assets

* Scene - separation of the Game based on the scenario. Our room can be made into one scene, more objects in the scene, more time to load and things to load into the memory.
* Parent - Child - as mentioned earlier if you move parent, you move the child as well. If you disable/hide the parent, same happens for the child. If you remove/destroy parent, child goes as well.
* Scene hierarchy - just like folder hierarchy.
* Assets - Sounds, Texture, 3D models, Shaders, Materials...everything that is a part of the game can be called an asset.

## MonoBehavior Basics

* Update - when you inherit from MonoBehaviour and implement method **Update()** that same method will get called 30-60 times per second. It may go even beyond 60 times, depending on settings and performance. Every logic that is time dependant goes there
* FixedUpdate - same as Update, but Unity engine will try to guarantee it will be called exactly 60 times per second. Mostly used when updating physic dependant stuff.
* LateUpdate - Let's say I move and you want to follow me. In my **Update()** method I will move with 5meters every frame. Your purpose is to wait for my current step at the current frame and then move at the next frame after me. Well this is LateUpdate - it will happen just after **Update()**
* Awake - it will get called upon scene start even if the script attached is disabled
* Start - it will get called after Awake only if the script and gameobject is active/enabled
* OnEnable/OnDisable - as you guessed called upon enabling/disabling of the GO (GameObject)
* Transform Component/Class - Empty GO will always have Transform component added. Transform is the component responsible for the position/rotation/scaling of the GO.
  + **Translate()** - Transform.Translate(X,Y,Z) - move the GO
  + **Rotate()** - Transform.Rotate(X,Y,Z) - rotate to GO
  + Activate/Deactivate - GameObject.SetActive(true/false) - if an GO is not active, then Update/FixedUpdate/LateUpdate won't get called
  + Destroy - Remove it from the scene