# GameObject Scripting

## Tag, Layers and Sorting Layer

* Tags - a way to mark objects and group them by some quality.
* Layers - Cameras render specific layers, Lights are illuminating specific layers, Raycasts are selectively hitting layers.
* Sorting Layer - Determine the order of things are drawn in the 2D mode

## Prefab

* Prefab is a way to save GameObject (and all of its components and settings) outside the scene.
  + Outside the scene means in the projects folder.
* Prefab can be anything you need to re-use in the game like bullets, planes, cars...
* Making prefab is super easy once you are in the scene just drag the gameobject to the project folder.

## Resources

* ProjectFolder/Assets/Resources is a folder in which whatever you put in will (simply said) get in the build. If you put a prefab called "Bullet", later you can load it by writing Resources.Load("Bullet"). Keep in mind you should put things in Resources only if you are going to use them.
* Pre-Load assets vs Resources.Load
  + Pre-loading assets means to keep them in the scene, then the scene file will be bigger and it will take more time to load. On the other hand if you make an empty scene and add all needed things using Resources.Load you will introduce CPU spikes when adding them in the Start() method.
* You should think about which things you need to have in the scene from the beginning and which not.

## Import Settings

* What is Mesh?
  + Mesh is a set of polygons that made up a model (3d/2d shape)
* Prefab vs Model
  + Prefab is GameObject + All Components one of which is MeshFilter that uses the Mesh to visualize the model.

## Unity Import capabilities

* Textures - Remember that there is two size of a texute/image when it gets imported. First is the size of the file in the disk, second that is used by Unity is the size of the image when it gets loaded in the memory. This size depends on the platform and the type of compression used. There is a lot of compression, usually Unity uses the most suitable. Always try to have the resolution of the images to be power of 2. This way it will get compressed with highest ratio.
* Models - After the 3D artist is done, the file can be imported in Unity by simply dragging it in. Most common softwares such as 3DMax, Maya, Blender are supported. See the Unity documentation for full list. There is one rule, if possible export your model to .fbx file, since this is a common most used format. All 3D softwares are capable to export to FBX. Most of the settings in the 3D Model importer are artist specific. Most important the relatable to a Unity Developer are
  + Scale Factor - you set the Factor or check to use the one that comes with the file.
  + Mesh Compression - reduces the quality but helps performance
  + Generate Colliders - instead of using Unity integrated colliders it will generate mesh collider. My advice is to use unity colliders to achieve the needed shape, it will be cheaper.
  + Rig - if the model is rigged by the 3D software whether to import the RIG information. Rigged model means it has "bones" structure to ease the making of animations.
  + Animations - whether to import any animation coming with the FBX file. You can adjust which part if the animations or to add specific animation events that will be triggered at chosen point in time of the animation.
  + Materials - if you want unity to import the materials as well.

## Common Scripting methods that will be used in this lab

* Attach Component - **AddComponent<Type>** where Type should inherit MonoBehaviour
* Get Component - **GetComponent<Type>**, if there is no such component it will return null
* **Camera.ScreenPointToWorld** - Display has two axes X and Y. If we touch a point in the screen we can find its 3D representation by saying how deep in the 3D space we want to reach. Camera.ScreenPointToWorld returns that point in the 3D space.
* **Vector3.Lerp** - If we have to points in the 3D space and we need the middle b/n them we will type - Vector3 **middlePoint = Vector3.Lerp(pointA, pointB, 0.5f);**
* If we want the get closer to pointB then last parameter will be bigger where 1f will means pointB.
* **Random.Range** - Random number b/n a range.

## Common GameObject properties

* **ActiveInHierarchy** - if gameObject is active but its parent is not active and we check if gameObject.activeSelf is true it will return true, but it won't be visile because its parent is disabled. ActiveInHierarchy will check whether an GameObject is really active/visible
* **activeSelf** - if GameObject is active or not
* **SendMessage** - SendMessage("MethodName"), it will search in the script attached to the GameObject for a public method called this name and execute it. Not really cheap method, but can be used if needed.
* **SetActive** - to set a GameObject to be active or not
* **Find** - searches for a gameObject with a given name, it will find it if it is active, if non found it will return null.
* **FindGameObjectWithTag** - same as above, but filters by Tag and returns array of gameobjects
* **Destroy** - marks the GameObject for destruction and will actually get destroyed in the next frame. If you want to destroy at the same frame - **DestroyImmediately**. Most cases **Destroy** will be enough.

## Input Manager

* Unity Mapping and a way to access the input

## Common Input class methods

* **Input.GetKey(KeyCode.A);**
* **Input.GetAxis("Axis-Name"); - axes returns from -1 to 1**
* **Input.mousePosition**
* **Input.GetMouseButton(index)**
* Always keep in mind there is 3 state of a press or touch - **Down, Hold, Up**
* **Input.GetKeyDown, GetKey and GetKeyUp**
* for touch screen - **Input.GetTouch() or Input.touches[0]**, always check before that if there is any touch at all using **Input.touchCount**.
* Under mobile **Input.GetMouseButton(0)** will also be counted as touch and will return true
* **Event.current** - another way to access current input - Event class contains properties like isMouse, charachter etc... For ex. user presses something and you need to detect that and get what exactly did he/she pressed - **Event.current.charachter**, you can do the same using Input class too

## Platform Dependent Compilation

We have few options when it comes to coding a game that will run on few platforms

1. All code is compiled, but only the one we need per platform is executed

**if(Application.IsEditor)**

**//your Editor code here**

**else**

**// rest of platforms**

2. Only the right code is compiled, but editing is a bit more complicated

**#if UNITY\_EDITOR**

**// your code here**

**#else**

**//rest of the platforms**

note - in mode two the code in the #else will get greyed out depending on the current chosen platform.

## Deployment

* **Standalone** - straight forward way - just hit Build and voila
* **Android** - You need to download Android SDK, you don't need the Android Studio or Eclipse etc...Recently google made their sdkmanager.exe to be only command based to enforce their IDE, but this won't stop us :D
  + sdkmanager is used to download the SDKs for different android OS version. Commands are simple and there is a lot of copy & paste examples
  + when you have the SDKs(usually the last one will be enough to have) then you need to point to Unity where it is. This is happening in Edit->Preferences->External Tools
  + after this you need the java SDK, from the same menu (External Tools) just press download and it will take you to the download page
  + You are ready to build - .apk file is generated which can be copied/installed on a phone or uploaded to your Google Play Developer Account
  + For a final release you'll need also to sign the build. In the Player Settings there is a menu in the bottom part where with the help of a wizard you generate the signature.
* **iOS** 
  + You'll need to download XCode prior to building
  + After that when you build in Unity an XCode project will be generated. Think of it as a Visual Studio Project.
  + After that you open the Xcode project and usually all certificates are automatically set with some exceptions
  + All needed certificates are managed on the iTunes Developer website and then downloaded onto your MacBook/OS X
  + Managing the certificates and so called provision profiles are a matter of following few tutorials on their website, they are setup only once.