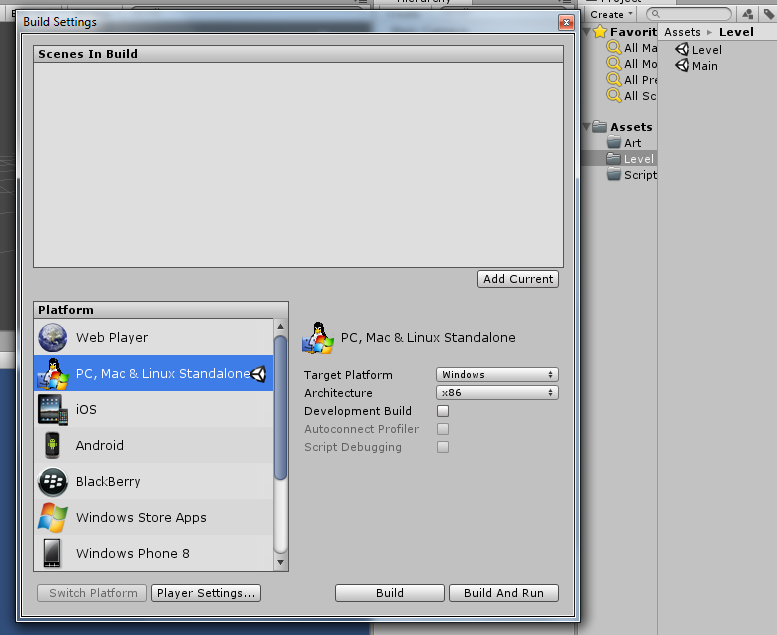
# Unity 101

Unity is a powerful tool to build no only 3D games but other application that you can think of (Unity 4.3 and above lets you create 2D games easily now!). The programming style would be component-based which means that you should create an object by adding components into them (e.g. a car object should have: a wheel component, body component, etc.).

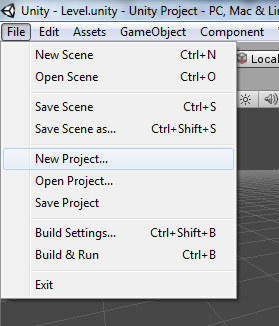
Use the official document for any components that you wish to understand better.  
<http://unity3d.com/learn/documentation>

Each level is a scene and before you publish your games, make sure you have added them into the build. (Do not leave it empty like in the picture!)

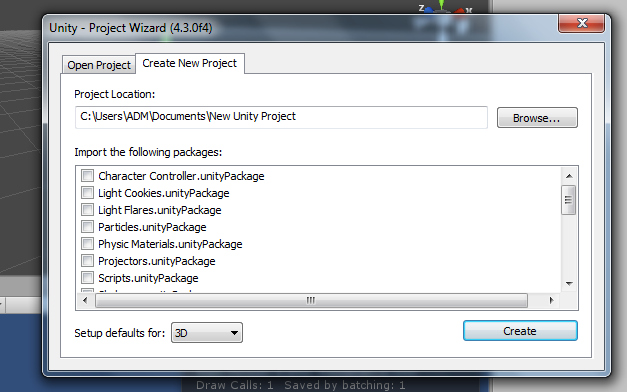


# Creating New Project

In order to create a new project go to file > new project.



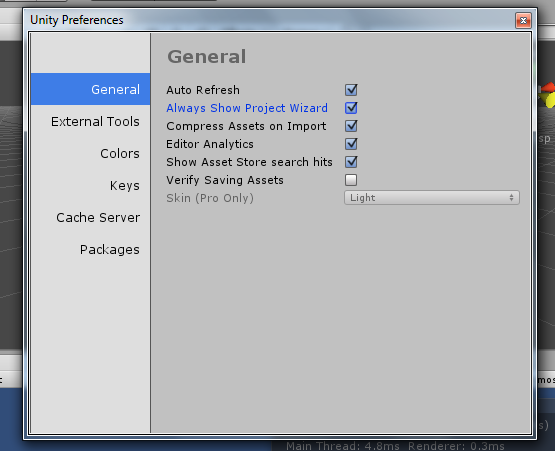
Uncheck everything that you will not be using (in this case since you are creating a new project from scratch we will not be importing any additional packages). Select browse and create a new empty folder. This folder will be where your project is located.



Create subfolder in the asset folder by right clicking in the Project Hierarchy > Create > Folders or going to the project folder itself and creating them inside the asset folder. (You may want to create “Resources” in your assets as it allows to use a Unity Function Resource.Load(); to load objects or textures using that function)

## Additional Tips

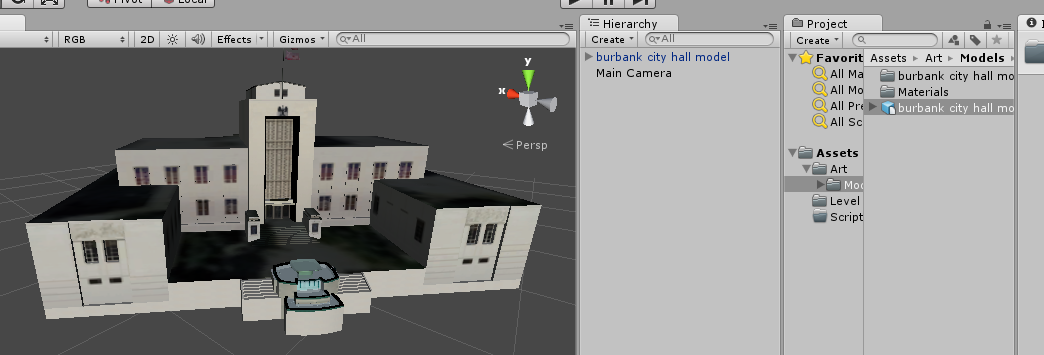
Go to Edit > Preferences and in the General Tap enable “Always Show Project Wizard” this will allow you to open up two or more unity projects concurrently which can be helpful if you wish to “copy” some values over from another project or using them as a reference.



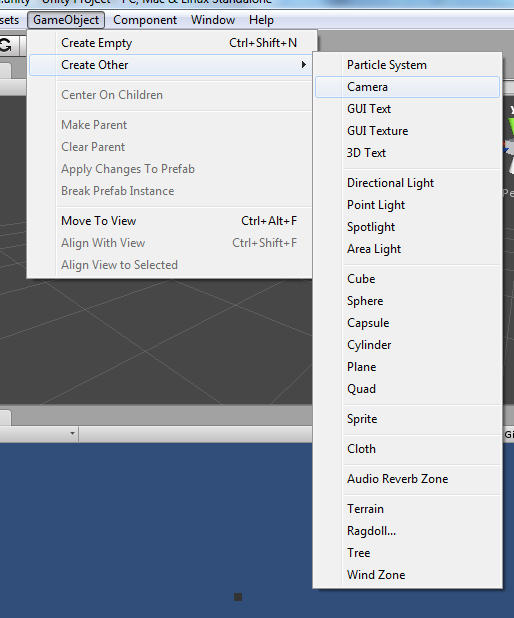
# Creating game objects

A game object can be empty, a 3D mesh, a 2D texture or even event triggers. You can import your own 3D objects that you have created in any external tools (Autodesk Maya, 3DS Max, Blender, Google SketchUp).

Drag and drop your model (.fbx, .obj, .ma) into the appropriate and Unity will import them automatically for you.



Or you can just create primitive shapes (cubes, spheres, cones) which will be much faster than trying to find free art assets. (E.g. GameObject > Create Other > Cube)



# Programming Style

Before you even touch any coding, figure out what kind of style you would like to adapt (Hungarian notation). It would be best to create a list of prefix, or suffix that you would like to have. (Here is an example used in the project)

|  |  |
| --- | --- |
| Datatype | Prefix |
| References | \_ |
| Class Variables | m |

You may also choose to give prefixes to other data types as well, but it all depends on yourself.

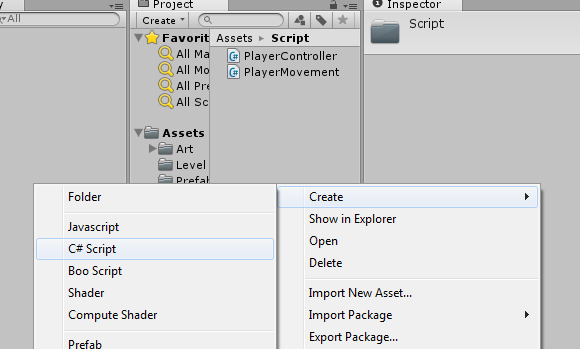
|  |  |
| --- | --- |
| int | n |
| string | str |
| Game Object | go |
| list | a |
| Bool/Boolean (if you’re a Java person) | b |
| Enum | e |

It is also a good practice to comment your code and also writing a short summary of what the code is used for. This will allow others (and yourself in the future) to know what the class is all about or how a certain code is supposed to function.

# Player Controller

The PlayerController Script will be used to control all the components that a player should have (for our game it will only be the movement, but you could add things like shooting and any ideas that you can come out with).

First create a script called PlayerController and another called PlayerMovement.



In the PlayerController Class add the following above the class name itself: [RequireComponent (typeof(PlayerMovement))]

This makes the PlayerMovement Script a needed component in whatever object the PlayerController script is attached to. (This way it will prevent you from making a mistake of not added the correct components that you need)

## Singleton

A Singleton class is a class that is the ONLY one in the entire scene. For our PlayerController we will make it a Singleton as there can only be 1 player at a time (even multiplayer games will have only 1 player, because you do not control the other players).

Type the following code (it is a little small but you could copy pasta it):

#region Singleton

private static PlayerController mInstance; // the Player Controller Instance

// get our Player Controller Instance   
 public static PlayerController GetInstance()

{

if(mInstance == null) InitSingleton();  
 return mInstance;

}

// Init Func for our Singleton Class  
 private static void InitSingleton()  
 {

if(mInstance == null) GameObject.Find("Player").GetComponent<PlayerController>();  
 else throw new System.AccessViolationException("Only 1 Player is Allowed");

}

#endregion

When we want to use any function (which you will see later) in our PlayerController Class we will call GetInstance(), and it will return us an instance of it (which will be the PlayerController).

The InitSingleton() function is used to set our instance to the object that can be found in the world.