INTRODUCTION TO UNITY

What is Unity?

Unity is a fully-fledged game engine with a built-in drag and drop editor. Its main focus is 3D games, but you can also build 2D games with it, and lately there have been multiple improvements in this area.

Unity runs on Mac and Windows, and it has an experimental Linux editor that is not as stable as the others. Unity's recent versions only support 64bit versions of windows, so if you are running a 32bit version of the OS, you'll have to pick up an older version.

How Does Unity Work?

Unity allows you to program your game using two programming languages: C# and UnityScript

C#

C# is the best language to use with Unity as of right now. Most tutorials and resources use it, and it is simply better than the other option. Unity supports C# 4.0 and .Net* 3.5. C# 6.0 and .Net* 4.6 are in an experimental phase. For those of you who don't know, **.Net is a software framework developed by Microsoft. Specifically in Unity, it allows you to use System namespaces (libraries) among other things. You can read more about it here and here.**

UNITYSCRIPT

Usually called JavaScript (in a Unity-specific context), UnityScript is a custom language with JavaScript-like syntax. Although it can do a lot, there are some things that it can't **compared to C#. This language is getting slowly deprecated in the latest unity versions, so it is recommended to avoid it altogether.**

Scripts written in these languages can communicate between each other, but it is not recommended. It's best that you pick a language and stick to it. The best choice **is C#, since UnityScript is getting deprecated.**

For all you technical artists, Unity uses HLSL and Cg for its shaders with some engine-specific adaptations. Unlike Unreal Engine, Unity does not have a node-based Material editor, so your only options are to write shaders yourself or buy one from the asset store.

HOW GAMES WORK IN UNITY

Your game is composed by Scenes (Levels). Each scene contains GameObjects, which are the "things" that populate your level. – **such as your player, your enemy, your terrain. Each GameObject contains Objects, also called components. These components are scripts that were programmed in order to do something specific, and then attached to a GameObject.**

Pros and Cons

PROS

Deployment – Unity exports to pretty much every possible platform: Mobile, Consoles, PC, VR, you name it, with minor porting and platform specific features needed.

Take a look at the full export list here.

Flexible – Unity is not limited to a specific type of game. Although 2D games are harder and less intuitive to make with the engine, everything is possible if you've got the **technical capabilities.**

Ease of Use – compared to other engines, such as Unreal and CryEngine, Unity's **learning curve is rather low and the engine can be picked up by anyone, although it is recommended that they have some prior knowledge in programming and/or game making.**

Community – Unity has one of the largest – if not the largest – community when it comes to game engines. There are resources all over the web, ranging from simple beginner-friendly tutorials to more advanced and complete tutorials. If you ever stumble across a problem you can't solve, there are tonnes of places online where you can ask. **See resources for links.**

CONS

If you're planning on making a small mobile game, Unity probably isn't what you're **looking for – its build (app) sizes are huge. An empty scene in Unity results in a 10Mb mobile application.**

Unity's source code is not available to the public. If you find an **engine bug, the only thing you can do is submit a bug report and pray that it gets fixed. It is possible to** license Unity's source code, but chances are it is unnecessary for your needs and **extremely hard.**

If you make more than $100k gross per year, you need to purchase a Unity Plus license (Up to $200k gross) or a Unity Pro license (Unlimited gross).

Unlike similar engines, such as Unreal Engine, there are many more advanced features that don't come out of the box, mostly regarding graphics and post**-fx. You need to either make these yourself, or you can head on over to the Asset Store and look for an already made solution.**

If you want to export to iOS or OSX, you need to xCode to compile your game, which can only run on a Mac. (Hackintoshes may or may not work, depending on your machine).

Installing and Using Unity

You can download Unity here.

It is recommended to have at least 15GB of free space in your hard drive, since Unity quickly takes up a lot of room in your HDD.

Unity comes with an IDE (code editor) that works out of the box called MonoBehaviour. If you're not expecting to do a lot of programming work, it works fine, but if you're coding most of the time you'll soon want a better work environment.

If your computer does not have a lot of RAM or processing power, i recommend that you check out Visual Studio Community. It is an open source IDE developed that comes

with everything you'll need. It's a simpler version o**f Visual Studio, and runs fine on lower end machines.**

If you're using a powerful machine, i advise you to check out Visual Studio Community. It's a much more powerful version of VS Code, and it's what lots of people use. It **requires a lot of HDD space (~10GB), so make sure you have enough.**

Take a look at it here.(Scroll down, it's in the bottom of the page).

To get to know Unity's interface and how its editor functions, take a look at the **resources for this section.**

Programming in Unity

When it comes to learning game engines, there is not a definite path to go down.

Some people like learning the programming languages this way, some people like learning that way.

Some like following tutorials and not focus so much on code. Some don't even code at **all and either use a Visual Scripting tool or hire people to do it for them.**

Everyone says their way is the best, but the truth is that everyone learns differently – **what works for me might not work for you.**

There are so many ways that you can learn and get started with programming in Unity that i cannot cover all of them. I can, however, provide simple explanations to what i believe are the most effective methods.
These methods are generic, and can probably be used in pretty much every engine you'll be using, not just unity.

METHOD 1

Getting familiar with the engine and learning as you go to suit your needs.

With this method, you won't be learning the programming language specifically. You will **be building your game and learning how to write the code for your it as you go. Need feature X? Search for implementations on Google, Youtube, StackOverflow, ask on forums, until you find a solution. Lots of times, people will give you advice on how you can improve your programming and that will help maintaining a clean and easy to use environment.**
This method will often lead to writing "ugly code" – since you're not learning the **programming language and its best practices, chances are there is a more effective solution t**o the problem you're facing.

METHOD 2

Learning the programming language you're going to use and applying it in the engine**-specific environment.**

In Unity, it's recommended that you use C#. With this method, you'd see yourself using Microsoft's tools to lear**n the language, its best practices and most used conventions and then bringing that C# knowledge over to Unity and adapting it to fit the engine. This method is the most difficult and is often considered scary, because it has you learn the language and then it has you learn how to adapt said language to the engine, giving you** a larger amount of work compared to the one you'd have using the other methods **described here.**

That being said, if you have never learned a Computer Language before, this may be a better route for you. Sometimes tutorials that pair C# and Unity can rush through the basics, which will be confusing if you don't know programming fundamentals.

METHOD 3

Learning the language in the context of the engine you're using.

This was the method that i personally used to learn C# and Unity – it helped me to know both the language and the engine-specific adaptations. Although i had some prior contact with C#, this method does not require it. Pick up a C# Unity tutorial and learn how the code functions, its main elements and how those elements and those functions

interact with the environment it's in. As time goes by, you'll see that you picked up **enough knowledge of the language to use it in another framework and that you already know how to handle it within the engine.**

Making your first game

One of the biggest issues beginners make when using unity is setting unreasonable scopes. I've seen many cases where people who just started out go out and make a "team" to make the next GTA6. Let's think for a seco**nd here – GTA5 was a multi-million dollar game with hundreds of people working on it simultaneously.**

There's no way a team of beginners can deliver a product of such quality.

Therefore, you need to control your scope. Your first project must be something simple – like a side scroller. With that first project, you can learn about movement, collision detection, life systems, and many other little things that will be helpful for the next game you make.

What matters most if that you learn new things. Although GameDev is a continuous learning cycle, it's very important that when you're starting out you try and do things you haven't done before and leave your comfort zone, so that you can learn how they are done and expand your programming knowledge. You'll see th**is will quickly become useful as you start making bigger and bigger games.**

And also, your first game will suck. A lot. It is normal, and it happens to everyone.

But don't worry. You're learning. You're absorbing information so that you can use it in **later** projects. Your next few games will also suck, of course, but with time you'll see **that the knowledge you got from your previous endeavours will allow you to finally achieve a good result.**

Scope Small, Constantly Learn, Never Quit.

Almost all the good resources for building games are in the previous section's resources, so you may want to start with one of those. This section's resources will be **more abstract game development techniques to help with your direction and creativity.**

Asset Store

One of Unity's greatest features is its Asset Store – **a marketplace where developers can submit their assets and put them up for sale. All packages are verified by unity to** make sure they're legit, and the platform is run by Unity themselves.

There are so many possibilities – from node-based Material Editors to 3D Model Generators to Low Poly Water shaders.

Not everything is paid though, there are some great free assets in the store as well!

If you want, you can also become a publisher and submit your own assets. Unity gives you 70% of your sales every month. It's a good source of revenue if you know your way **around the engine!**

Releasing a Game

Alright, so you have the world's most lit game. What do you do now? You release it of course! Video games are an artform, and even if some people don't believe that to be **true, you should want others to play your games!**

There are many places where you can release your projects. Some of them are more complicated to get onto that others (Steam versus itch.io), but once you release your game, people you have never met before will be able to play it and give you valuable feedback.

The resources here will include places where you can release your game and also articles about how to get people to play them.