**The Unity Game Engine. (From: A History of the Unity Game Engine.)**

**Abstract**

Unity (commonly known as Unity3D) is a game engine and integrated development environment (IDE) for creating interactive media, typically video games. As CEO David Helgason put it, Unity “is a toolset used to build games, and it’s the technology that executes the graphics, the audio, the physics, the interactions, and the networking.” Unity is famous for its fast prototyping capabilities and large number of publishing targets.

The first version of Unity (1.0.0) was created by colleagues: David Helgason, Joachim Ante and Nicholas Francis in Denmark. The initial product was launched on June 6, 2005. The goal was to create an affordable game engine with professional tools for amateur game developers while “democratizing the game development” industry. The three were inspired by the easy workflow, simple asset pipeline, and drag-and-drop interface of Apple’s Final Cut Pro product. When originally released, Unity was available solely for Mac OS X, and developers could only deploy their creations to a few platforms. The current version (4.3.1 as of this writing) is supported on both Windows and Mac OS X, and offers at least a dozen target platforms. Publishing 64-bit executables has been possible in Unity for a long time, but a 64-bit version of the Editor is not yet available, though highly anticipated from the development community.

**Asset Pipeline and Workflow**

Unity’s asset pipeline is leading the industry in ease of use, and compatibility with current asset development software. To import an asset into Unity, the user simply has to drag it into project window and Unity takes care of the rest. When editing an asset, Unity automatically updates the asset with the latest modifications and spreads them across the entire project.

Unity supports importing from many applications (see appendix for complete list of supported file types), but only natively supports Autodesk FBX and COLLADA for models with bones or animations. When importing most models of this type, Unity opens the original program that created the model and exports it to a FBX file that Unity can use. This is performed automatically and is completely behind the scenes. It saves time that developers would otherwise be spending trying to convert assets into the proper format. If the game developer needs even more precise control over the import settings, they can extend the editor to fit their needs using the Unity Editor API namespace. Unity’s workflow also allows developers, who are targeting multiple platforms, to use the same source files by allowing the user to override the import settings for each platform targeted. If the developer is working on a massive project with many assets, they may have trouble locating the correct one in a timely manner. The project window allows the user to search and locate objects very efficiently using many parameters, including user-defined tags. The editor also includes a preview window, enabling the developer to view the appropriate asset without having to import it into the scene or apply it to an object.

With Unity 3.4, support for algorithmic substances was added. Substances are highly customizable, dynamic and procedural textures that can be manipulated in real-time to produce unique effects. They have an extremely small file size, which makes them ideal for streaming. They are dynamically generated, so they can be scaled up without losing quality. For platforms which do not support this technology yet (e.g. iPhone, Android), Unity can bake substances into a standard bitmap.

**Non-Game Uses**

Recently, developers are beginning to realize that game engines can be successfully used for non-game applications. For example, architects can easily prototype ideas, artists can create interactive art installations, or researchers can use them for data visualization. The inverse does not apply, however. It is practically impossible to create a decent game using a computer-aided design (CAD) program. In an interview, Francis suggested that this, along with Unity’s ease of use and gentle learning curve, is one of the main reasons Unity has flourished. In a poll that was given to Unity customers, it was discovered that a third of them were not using Unity to make games.

**Multiplatform**

Another reason Unity has become so popular is its ability to deploy to a wide variety of target platforms, using the same code and assets. It takes two button clicks to get your game running on another supported platform. One to switch platforms, and the other to build and run. Unity currently supports four main categories: Mobile, Desktop, Web and Consoles.

**UnityScript**

UnityScript is a Javascript-like language, and the better choice for novice users. The majority of Unity developers use UnityScript, so there are more examples, and it is easier to get assistance on IRQ or the Unity forum. UnityScript is easy to learn and fast to type. It handles a lot of type casting behind the scenes and allows the user to switch between dynamic or strict typing. Javascript is a prototypal language, whereas UnityScript is a classical one, and therefore receives the benefit of classes and inheritance. Even though UnityScript supports dynamic typing, it can be more inefficient than C# because the compiler has to take into account that an object’s type may change. Privacy and private variable creation work the same way as in C#: all the programmer has to do in put the keyword “private” before they declare something. Although generally considered the beginners language, there are a few well-known developers who prefer UnityScript even after learning the other available languages.

**C# -** C# (pronounced C-Sharp) is moderately more challenging to learn (and a lot more to type) than UnityScript, but allows the programmer to have complete and precise control. C# does very little “behind the scenes” for you, and requires the developer to manage most of the components manually.

**Unity Cloud**

Unity Cloud is an upcoming set of tools designed for Unity developers “to build, market and operate compelling multiplatform games without an extensive investment in cloud infrastructure.” Developers will be able to select as many or as few of the cloud tools as they desire. The first version of the Unity Cloud service will reportedly support: player locating, title promotion, and cross-platform advertisements. All of these features will be built-in, without the need for a third party library. Developers will be able to use the Unity Cloud ad service to earn extra revenue, or to cross-promote their other applications. “In short, we are working to unlock the reach of over half a billion Unity mobile game installs to create the largest game promotion network on the planet”.

**Asset Store**

On November 10, 2010, Unity Technologies announced and launched a third-party marketplace for Unity developers called the Unity Asset Store. This market is built right into the IDE, so that developers can browse and purchase assets for their application without ever leaving their project. Individual developers can also post assets and packages. This provides the Unity community with an entirely new method to monetize their skills and talents. Unity Technologies keeps 30% of revenue and the asset designer receives the remaining 70%. Top selling assets produce enough money for a person to comfortably live on. Calle Lundgren, from the Indie game company VisionPunk, has made enough money selling his assets that he has lived for more than two years on nothing but these earnings. Many developers also use the Asset Store to fund their more advantageous projects on the side. Currently, there are thousands of asset packages available in the store, with more being added every day.

**The Unity Editor -** The Unity editor is comprised of many sub-windows. The most commonly used are: Project Browser, Inspector, Game View, Scene View, and Hierarchy.

**Project Browser -** The Project Browser is the window that contains all the assets that have been imported into Unity and are available for use. The layout is almost identical to Finder in Mac OS X, and Explorer in Windows. This makes developers feel more familiar and confortable by providing an interface they already use on a daily basis.

**Inspector -** The Inspector window is where the details of every GameObject are viewed and modified. This is where the developer can tweak values to get just the right feel for their game. The Inspector shows all the Components that are attached to an object (e.g. Scripts, Physics, Colliders, Sound). This is also where variables exposed from scripts can be assigned or changed.

**Game View -** The Game View provides the user with a WYSIWYG (what you see is what you get) preview of what their game will look like when they build it. It includes full input and permits the developer to test their changes without having to wait for their project to be compiled and deployed on the target platform.

**Scene View** - The Scene View is where the game is constructed. The developer can drag and drop assets from the project view. Familiar 3D handle controls and grid snapping allows the user to place their objects in the perfect position, down to the pixel.

**Hierarchy** - The Hierarchy window contains a list of all the objects that are in the current scene. This list is automatically updated when an object is brought into the scene. In this window, by dragging an object on top of another object, the developer can assign parents or children.