

Intermediate Report

Reducing the workload in Unity

Guillem Poy Trujillo

Treball de Fi de Grau 2020-2021

TABLE OF CONTENTS

| | |
|--|-----------|
| ABSTRACT | 1 |
| INTRODUCTION | 2 |
| OBJECTIVES | 3 |
| Reducing the amount of work in most Unity projects | 3 |
| Finding the best metrics to evaluate the value of each feature | 3 |
| Secondary objectives | 3 |
| Usability | 3 |
| Good software design practices | 3 |
| Demos | 4 |
| REFERENCES | 5 |
| THEORETICAL FRAMEWORK | 9 |
| How to choose which features to implement | 9 |
| Calculating the payoff and the investment | 10 |
| Unity Software | 12 |
| Unity guidelines | 12 |
| Good Software Design Practices | 22 |
| SCHEDULE AND METHODOLOGY | 23 |
| PROJECT DEVELOPMENT | 25 |
| 1. Finding weaknesses in the existing asset | 25 |
| 2. Finding the most common needs | 27 |
| 3. Deciding the metrics | 29 |
| Payoff | 29 |
| Difficulty | 30 |
| 4. Rating the features | 31 |
| 5. Feature implementation | 34 |

| | |
|---|-----------|
| Fixing the weak points | 34 |
| Automatic application of asset's recommended settings | 34 |
| The DebugPro class shouldn't be needed | 36 |
| Improving the Pool class | 37 |
| Improving SimpleAnimation | 39 |
| Increasing the usability of the EasyRandom class | 41 |
| Increasing the usability of the shortcut "Clear Console" | 42 |
| Improving organization of the Essentials file menu entries | 44 |
| Implementing new features | 45 |
| A way to animate Components (camera, transform, ...) without needing an animator controller | 45 |
| C# classes integrating common needs (such as a "rotator" that could rotate between different outputs every time a new one is requested) | 47 |
| A way to snap an object to the surface of any other's mesh without leaving any distance between their meshes | 49 |
| Default existing presets for components like the camera, canvas, ... | 51 |
| Extensions for components and classes like the Transform or the Vector | 55 |
| Integrating externally developed features | 57 |
| Publishing to the Asset Store | 58 |
| Changes on each version | 60 |
| REFERENCES | 61 |

ABSTRACT

English:

This final year project tries to identify and implement the most useful additions that could be integrated into Unity in a reasonable time frame by one person with the objective of reducing as much as possible the workload of the users of the engine.

Spanish:

Este proyecto de fin de carrera trata de identificar e implementar las adiciones más útiles que podrían integrarse en Unity en un plazo razonable por una sola persona con el objetivo de reducir al máximo la carga de trabajo de los usuarios del motor.

Catalan:

Aquest projecte de fi de carrera intenta identificar i implementar les addicions més útils que podrien integrar-se a Unity en un termini raonable de temps per una sola persona amb l'objectiu de reduir al màxim la càrrega de treball dels usuaris del motor.

INTRODUCTION

Back in January 2020, the idea of creating an asset that added most of the features that had been found missing in Unity during the development of multiple projects such as "Drink & Play" and multiple university projects was born.

That asset included some of the most commonly needed features across the projects such as:

- Automatic installation of the Quick Search package
- Ability to disable the Warning C60649 - which is very common in Unity projects due the fact that variables can be defined through code but initialized in the inspector
- Improved Debug class to be able to debug IEnumerable objects.
- A class to be able to create pools easily
- A way to animate Transform and RectTransform components through the inspector and scene objects
- A class to generate random numbers with self-explanatory methods and parameters in addition to some ways of generating random results typically used in such as pseudo-randomness
- Extensions for the classes Component, Float, GameObject, ICollection, IEnumerable, Int, LayerMask, RectTransform, String, Transform and Vector
- Additional shortcuts to clean the console, save the project and the scene at the same time, ...

After developing most of the features, the asset containing them was published in the Unity Asset Store¹ under the name of "Essentials". That asset can be found in the following link: <https://assetstore.unity.com/packages/slug/161141>

However, this asset only contemplated personal demands of features that were found missing in Unity. This was found very selfish by the developer itself, so the idea of updating the asset so it would implement features requested by the community was born.

In addition, the intent of making the features feasible for most of the projects made with Unity grew up. So, that was a thing that was wanted to be considered alongside the fact of making them feel as seamless as possible with the Unity platform and API.

¹ "Asset Store" and "Unity Asset Store" and "Store" might be used interchangeably referring to the "Unity Asset Store".

OBJECTIVES

Reducing the amount of work in most Unity projects

The main objective of this final year project is **to develop an asset that would work as a set of tools and features that the average developer would most likely need** in most of the projects made with Unity to reduce as much as possible the workload.

Finding the best metrics to evaluate the value of each feature

To achieve the main objective of developing an asset capable of reducing the workload in the majority of projects, **a protocol to find and prioritize needed features and tools** has to be designed and created.

This protocol should end up giving a value for each feature that would be used to sort the different features depending on how worth are they and would help to choose those that theoretically would have the most impact related to them.

Secondary objectives

The main goal of reducing the amount of work in most Unity projects is wanted to be achieved following secondary objectives that would increase the value of the asset:

Usability

Usability is the “extent to which a system, product or service can be used by specified users to achieve specified goals with **effectiveness**, **efficiency** and **satisfaction** in a specified context of use” (International Organization for Standardization [ISO], 2018, sec. 3.1.1).

To make the asset effective, efficient and satisfactory, it has been chosen to try to integrate it seamlessly in the Unity Software following their internal guidelines and imitating their tools, organization, ...

In addition, as many integrated tutorials, tooltips, help elements, ... would be used alongside the most self-explanatory naming found for everything.

Good software design practices

With the objective of facilitating as much as possible, the development, maintenance and upgradability of the asset, using the best coding practices as frequently as possible becomes a secondary objective.

Demos

In addition to the development of the asset, some demos are going to be developed.

Those demos are going to have multiple objectives:

- Proving that the asset can be used in different scenarios.
- Showing how to implement and use the asset in a Unity project.
- Show that, indeed, implementing the asset reduces the workload in numerous ways.
- Increasing the usability by having examples to learn from.

REFERENCES

Despite not having found an asset dedicated to integrate different types of features missing in the Unity platform, there are many assets that focus in just one feature but do some things pretty well and a lot can be learnt from them, so some of their features can be imitated in the developed asset:

- **Pooling:**
 - Multi Object Pooler: This asset allows the user to control multiple types of objects in the same pool and configuring the pool through the inspector. It can be found in the following link:
<https://assetstore.unity.com/packages/tools/integration/multi-object-pooler-130165>
 - Ultimate Pooling: It is a very user-friendly asset that allows the user to have a lot of control on their pools with things like the prewarming of the pool with the number of objects spawned per frame while keeping the simplicity and ease of use in mind. It can be found in the asset store using this link:
<https://assetstore.unity.com/packages/tools/ultimate-pooling-64950>
- **Component Animations:**
 - DOTween Pro: It allows the user to create animations for certain characteristics of certain components through the inspector without the need of using Unity's animation manager. However, it only allows you to do certain types of animations for specific elements. This asset can be found through this link:
<https://assetstore.unity.com/packages/tools/visual-scripting/dotween-pro-32416>
- **Increase of the ease of use:**
 - Playmaker: Makes programming way more user-friendly for non-specialized users by adding a visual scripting tool. This tool includes a guided tour to get familiarized with it, templates that give you useful parts of code that you will not need to create, high customization and extensive documentation. It is published in the Asset Store under this link:
<https://assetstore.unity.com/packages/tools/visual-scripting/playmaker-368>
 - Behavior Designer - Behavior Trees for Everyone: An easy way to create and manage behaviour trees with a visual editor instead of through code but imitating the API from "MonoBehaviour" that most users are already familiarized with. It can be found in the following link:
<https://assetstore.unity.com/packages/tools/visual-scripting/behavior-designer-behavior-trees-for-everyone-15277>

- Visual State Machine: This asset makes creating state machines very easy without using any code at all if not wanted. It follows the same aesthetics from the tools created by Unity Technologies like the Animator Controller so, interacting with it is straightforward if you already are used to Unity's integrated tools. This asset can be found through this link:
<https://assetstore.unity.com/packages/tools/visual-scripting/visual-state-machine-157252>
- Discourse: Allows to the users a way to create non-linear dialogues, cutscenes and manage events through a visual editor that tries to be as clear and easy to use as possible. Additionally, it supports localization and has invested to try to make the errors made by the user as solvable as possible with a full undo system. It is found in the following link:
<https://assetstore.unity.com/packages/tools/visual-scripting/discourse-146948>
- Magic Light Probes: This asset places light probes around a desired area without the need of any major intervention from the user. Doing so, the time dedicated to place the needed light probes is reduced by a huge amount. This tool can be found in the Asset Store under this link:
<https://assetstore.unity.com/packages/tools/utilities/magic-light-probes-157812>
- **Editor**
 - Odin - Inspector and Serializer: Allows the programmers to easily create custom windows and inspectors. It follows the same patterns as the default attributes provided by Unity to create custom inspectors, but it extends the functionality to a whole other level. The asset can be found here:
<https://assetstore.unity.com/packages/tools/utilities/odin-inspector-and-serializer-89041>
 - Asset Usage Finder: Even though it is a very useful tool to keep any project's assets usages under control, it is really impressive how they managed to give the wanted information and tools seamlessly just by the click of one button. The asset can be found using this link:
<https://assetstore.unity.com/packages/tools/utilities/asset-usage-finder-59997>
 - Enhanced Hierarchy 2.0: Provides and allows the edit of information (usually found in the console or in the inspector) regarding each GameObject in the Hierarchy, making it easier to work with. It can be found through this link:
<https://assetstore.unity.com/packages/tools/utilities/enhanced-hierarchy-2-0-44322>
 - Asset Cleaner PRO - Clean | Find References: This asset does a really well job of giving the most basic information in a very elegant and seamless way. With the tool activated, the needed data is provided and the finding of

unused assets is straightforward thanks to the color coding applied to the default Unity's Project window. It can be found in the Asset Store following this link:

<https://assetstore.unity.com/packages/tools/utilities/asset-cleaner-pro-clean-find-references-167990>

- **Scene tools**

- Grabbit - Editor Physics Transforms: This asset addresses the lack of any tool in Unity that allows the placement of objects in the scenes using physics. It makes it very seamlessly and without any extra work required by the user. This asset can be found through this Asset Store's link:
<https://assetstore.unity.com/packages/tools/utilities/grabbit-editor-physics-transforms-182328>

- **Commonalities**

Some assets provide elements that are common in a lot of video game mechanics and designs. Creating them in a generic way can reduce significantly the amount of work because the project programmer would only need to do the final tweaks instead of the whole system. Some examples can be found in the following assets with links:

- Inventory:
<https://assetstore.unity.com/packages/tools/utilities/inventory-96372>
- Third Person Controller - Basic Locomotion Template:
<https://assetstore.unity.com/packages/templates/systems/third-person-controller-basic-locomotion-template-59332>
- Camera Controller:
<https://assetstore.unity.com/packages/tools/camera/camera-controller-13768>

Additionally, hundreds of artistic assets can be found in the Unity Asset Store that can be used as reference or base for many projects' assets. They tend to have customization available to make each integration unique, but they all can be used directly out of the box. Some of the most liked and widely usable assets are:

- **VFX**

- Easy Performant Outline 2D | 3D (URP / HDRP and Built-in Renderer) v3.0:
<https://assetstore.unity.com/packages/vfx/shaders/fullscreen-camera-effects/easy-performant-outline-2d-3d-urp-hdrp-and-built-in-renderer-v3--157187>

- Stylized Water 2:
<https://assetstore.unity.com/packages/vfx/shaders/stylized-water-2-170386>
- Advanced Dissolve:
<https://assetstore.unity.com/packages/vfx/shaders/advanced-dissolve-111598>
- Lux Water:
<https://assetstore.unity.com/packages/vfx/shaders/lux-water-119244>
- **Animations**
 - Basic Motions PRO Pack:
<https://assetstore.unity.com/packages/3d/animations/basic-motions-pro-pack-157744>
 - RPG Character Mecanim Animation Pack:
<https://assetstore.unity.com/packages/3d/animations/rpg-character-mecanim-animation-pack-63772>
 - Mega Animations Pack:
<https://assetstore.unity.com/packages/3d/animations/mega-animations-pack-162341>
- **Textures and materials**
 - AllSky Free - 10 Sky / Skybox Set:
<https://assetstore.unity.com/packages/2d/textures-materials/sky/allsky-free-10-sky-skybox-set-146014>
 - Cartoon Town Materials Pack:
<https://assetstore.unity.com/packages/2d/textures-materials/cartoon-town-materials-pack-162934>
- **Models**
 - HQ FPS Weapons:
<https://assetstore.unity.com/packages/templates/systems/hq-fps-weapons-177306>
 - Stylized Nature Pack:
<https://assetstore.unity.com/packages/3d/environments/stylized-nature-pack-37457>

THEORETICAL FRAMEWORK

How to choose which features to implement

A way to prioritize features has to be selected so the work done ends up having the most significant value possible with the time and resources available for this project.

One of the best options might be the PICK chart. Originally developed by Lockheed Martin (George, 2003, p. 292-293) it “helps a team organize and prioritize its solution ideas by separating them into four categories: Possible, Implement, Challenge, or Kill” depending on their difficulty/investment and their payoff.

The idea behind it is that you should focus on those ideas with as much payoff as possible with less investment needed.

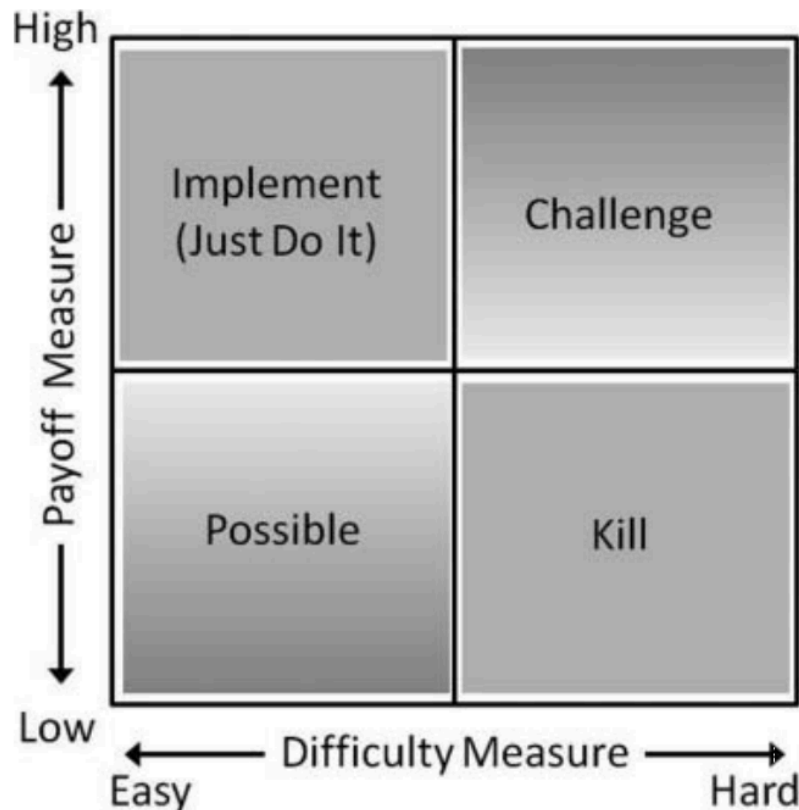


Figure 1. Basic layout of the PICK chart.

At the end, the best ideas with more value would be the ones that have the highest payoff for the lowest investment. In other words, the value (or efficiency) of each feature can be calculated by dividing the payoff between the investment as follows:

$$value = \frac{payoff}{investment}$$

Badiru & Thomas (2013, p. 6-8) express that the PICK chart can be used to evaluate the efficiency, effectiveness and productivity among other metrics in a very similar way that the one proposed to calculate the value of each feature. For instance, a way to calculate efficiency is:

$$efficiency = \frac{output}{input}$$

Calculating the payoff and the investment

As exposed by Badiru & Thomas (2013, p. 3) 'the "PICKing" technique is normally done subjectively by a team of decision makers through a group decision process. This can lead to bias and protracted debate of where each item belongs on the chart.'

However, some elements can help determine the difficulty and the payoff of each feature.

Moran & Riley (2014 p. 1) list some of the factors that could contribute to the success of each feature:

- Scope of control
- Scope of influence
- Cost
- Time
- Staff availability
- Available data
- Ease of gathering data

Other sites (TXM Lean Solutions, 2020) recommend asking a question to the team to help clarify the idea, its scope and difficulty such as:

- Have we done something similar before?
- Have we seen something similar at another company?
- Is this known technology?
- Can we implement it ourselves?
- Do we need to rely on another department to make this work?

At the end, no objective way to determine the difficulty and payoff of each feature has been found. So, to determine their values, the subjective knowledge and experience is going to be a main component during the use of the following approaches to determine the payoff and the difficulty of each one of them:

Payoff: Due to the fact that the main objective of the asset is to reduce the workload in Unity, in the end, what matters the most, is the time that each feature saves to the developer that, if it was missing, would have to implement it by himself or manually do the work.

Difficulty: This project is only going to be built up by one person, so no team issues have to be addressed, but it has to be considered that all the knowledge needed is going to be coming from previous projects or gathered during the development of each feature. So, at the end, the difficulty to develop each feature is going to be mainly related to the time of development and the gathering of the information needed.

Unity Software

According to Unity Technologies (2021) "Unity's real-time 3D development platform lets artists, designers and developers work together to create amazing immersive and interactive experiences".

Unity is one of the most popular Game Engines in 2021 (Schardon, 2021) and "has become a staple of the indie game industry". Additionally, she states:

The engine is not only well-suited for both 2D and 3D games of any type, but it is also a popular choice for VR and AR development as well, thanks to many companies and developers creating convenient SDKs for the engine. Beyond this, Unity also has a sizeable community, with an active Asset Store with both free and pay-to-use assets at your fingertips.

Additionally, Axon (2016) wrote that "according to Unity, more than 6 million registered developers use the platform, and 770 million gamers enjoy Unity-made titles" in 2016.

What is different about Unity?

As explained by Axon (2016), Unity Labs EVP Sylvio Drouin commented that some of that popularity might be thanks to the fact that before Unity, most of the game engines that existed were usually engines where you had to start coding in C++ and call APIs and build the scaffolding yourself. That made them really targeted at engineers that understand what they are doing, while Unity is very asset-driven.

Unity guidelines

It has not been found any document about what are the internal guidelines for Unity's API and tools.

So, instead, the asset will have to follow the "Submission Guidelines" made by Unity for any asset that wants to be published in the Unity Asset Store.

Those instructions do not only explain how the asset page in the store should be created or which contents to include but also how to name the folders, organize the editor extensions, ...

The most applicable guidelines for the content and its organization for the asset according to the Submission Guidelines by Unity in 2020 found in <https://unity3d.com/asset-store/sell-assets/submission-guidelines> might be the following ones:

Content Organization

- Packages should be nested under a folder with either the publisher name, or package name as the title, except for the folders outlined in the Special Folders and Script Compilation Order documentation².
- The assets should be properly sorted into folders with English titles depending on their type (Mesh/Script/Material/etc).

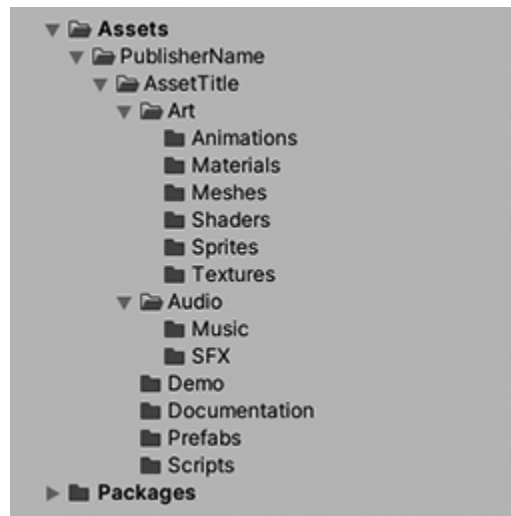


Figure 2. Example of an organized and properly named folders

- All duplicate, unused or redundant files must be removed from the project before submitting.
- All files must follow a consistent naming convention and their names need to represent the content they are providing.
- File paths for assets should be kept under 140 characters.

Documentation:

- All projects that require information on how to set up and properly utilize are required to have local documentation in English and should assume users have a basic understanding of the environment.
- Documentation file types must be in .txt, .pdf, .html or .rtf file types. In editor tutorials, inline documentation, and tooltips are appropriate as well.
- Shader documentation must explain each shader property. Do not assume that the property names alone are enough for the user to figure out what they do.
- If the product has contents to show off, they should be displayed in a demo scene. If the project includes a collection of assets, it must include a scene that showcases all of their assets laid out in a grid or continuous line.

² The documentation can be found in the URL:
<https://docs.unity3d.com/Manual/ScriptCompileOrderFolders.html>.

Files

- Preferences, settings or supplemental files for another Asset Store product must be nested in a .unitypackage file type.
- Files of type .zip or .rar can only be used for files that do not natively function in the Unity editor. (I.e. Blender, HTML Documentation, Visual Studio Projects, etc.)
- Make sure all the assets are as optimized as they can be or if applicable. The size limit for an asset for the Asset Store is 6GB.

Generic Art Content

- The package must hold a consistent artistic style.
- The content submitted should display a certain degree of professional design, construction and be ready to be used in a development pipeline.
- Mesh assets should be in files of type .fbx or .obj. Procedural or other types of meshes generated at runtime are acceptable as well.
- All visible meshes are required to have a paired set of textures and materials assigned to them. In addition to a corresponding prefab set up with all variations of the texture/mesh/material that is being provided and with a collider component that fitting the mesh.
- Large environments or scenes must be broken up into individual meshes.
- Prefabs must have their position/rotation set to zero, and the scale set to 1.
- All meshes must have a local pivot point positioned at the bottom center of the object, consistently in a corner of any modular objects, or where the object would logically pivot/rotate/animate.
- All meshes should be rotated to have their positive Z facing forward.
- Meshes need to be at a 1 Unit : 1 Meter scale.
- Assets must not have an unreasonably excessive mesh density. Additionally, photoscanned data must be retopologized and optimized. No meshes that are the direct result of scans are allowed.
- Assets must not have their UV's automatically unwrapped.

Rigs

- Character models must be weighted to an accompanying rig. The rig can either be set up with Unity's Mecanim system or can include your own animations.
- When set to animations, rigs should not show any obvious creases or unusual deformations.

Animations

- Bipedal animations should be ready to use with Unity's default "Humanoid" avatar.

- Projects of non-Bipedal animations need to include a demonstration mesh with a well documented rig breakdown so that users can create meshes designed for the animations.
- Animations need to be sliced and named. Single long animation clips are not allowed.
- Any animations that are developed from mocap data need to be cleaned up into sliced and usable animations. No raw mocap data is allowed.
- Animations should have fluid movement without any jarring transitions.
- All animation assets should have a video demonstration showcasing the included animations.
- Animations for unique characters should be contained in their own prefab.

Textures and Materials:

- Texture files need to be in a lossless compression format such as .png, .tga, or .psd.
- Any PBR³ package or that is using the Standard Shaders must include at least an albedo, normal, metallic (or specular) and smoothness texture map.
- Tileable textures must tile without any seams or obvious edges.
- Maps with an alpha channel need to be paired with a material that can read said alpha.
- Normal maps need to be marked as a "Normal Map" in the import settings.
- Content intended to be run on mobile or lower-end hardware should have atlased textures in order to reduce performance impact.
- Assets should not have an excessive number of materials assigned to them.
- SBSAR⁴ and other procedural materials need documentation or a demo scene showcasing the parameters included.
- Dimensions of textures should be pixel counts that are a power of 2 when appropriate.
- Materials should include all appropriate textures and be properly set up.

Sprites

- All sprite sheets must be imported with the "Sprite" import settings.
- Sprites need to be properly sliced and named using the import settings.
- Sprite animations need to be spliced, named and set up as proper clips before submission.

³ Information regarding how to work with Physically Based Rendering (PBR) can be found in the following blog post:
<https://blogs.unity3d.com/2015/02/18/working-with-physically-based-shading-a-practical-approach/>

⁴ SBSAR files are exported graphs from Substance Designer

GUI Packs

- Submissions of GUI packs must include a functional demonstration scene that showcases all of the components being usable.
- GUI components need to have their elements separated and named either before import or through our sprite editor settings.

Particle Systems

- Particle systems should be saved as a prefab to easily drag and drop into their scene.

Scripts

- Scripts should not throw any compiler or generic errors. Any potential errors must be properly caught and presented to the user through the debug log.
- All submitted code must be commented in English and readable with no spelling/grammatical errors.
- Functions and variables should be named appropriately.
- Scripts must include namespaces within which all named entities and identifiers must be declared.
- Assets that support Android builds should target 64 bit architecture.

Editor Extensions

- Editor extension file menus must be placed under an existing entry. If no existing menus are a good fit, you can place it under a custom menu titled "Tools" trying to keep the editor clean and organized.

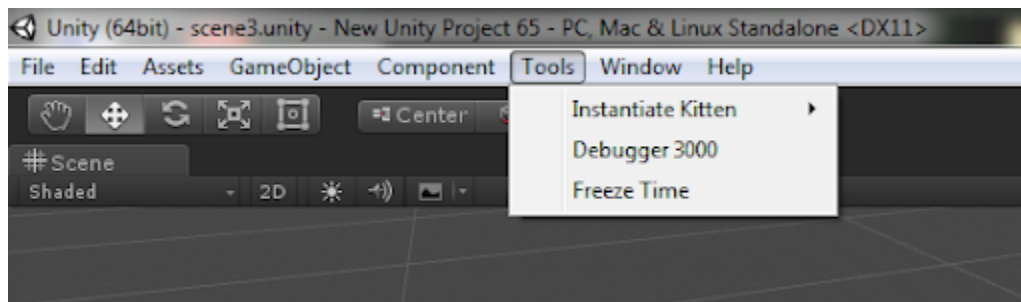


Figure 3. Example of an ideally nested Editor Extension

- Undo operations must be supported⁵.
- If a Server-Based Plugin is used, any new databases with necessary tables must be automatically populated.

⁵ Information regarding the "Undo Support" can be found in Unity documentation through the following link: <https://docs.unity3d.com/ScriptReference/Undo.html>

Complete & Template Projects

- Complete and Template Projects should be designed as instructional, tutorial or framework products.
- Complete project's documentation must include in-depth information about how the project is designed and how users can edit and expand on it, not only how to run the project.
- Products should offer some unique aspect to our development community and should not violate any copyright protections.

Audio

- Audio files must be normalized.
- Audio files must play properly inside the Unity inspector and supported Unity audio formats⁶ should be the only ones existing.
- Using a lossless format for audio files such as .wav files is strongly recommended.

To end, all content on the Asset Store is required to operate within the Unity Editor.

Additionally, a good practice would be to try to write the code structured and written in a way that tries to imitate as much as possible the default Unity's API coding conventions.

To do so, the following resources have been studied:

- Microsoft's C# Coding Conventions (C# Programming Guide):
<https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/inside-a-program/coding-conventions>
- Documentation by Taranov, Konstantin:
<https://github.com/ktaranov/naming-convention/blob/master/C%23%20Coding%20Standards%20and%20Naming%20Conventions.md>
- The public GitHub repositories of Unity: <https://github.com/Unity-Technologies>

After studying the previous resources, the following rules have been collected considering the relation between Unity and C#⁷:

⁶ For more information on supported formats, see Unity's Audio File Documentation in the following link: <https://docs.unity3d.com/Manual/AudioFiles.html>

⁷ Extended information regarding the rules and examples can be found in the following document: https://docs.google.com/document/d/1uD-zW-EXkl-iDXaCJ_Jt5BtdasZQeKCkTGIV_4RivIs/edit?usp=sharing

Layout

- You should declare a variable, method or function before it can be referenced by other components in the same script.
- Write only one statement per line.
- Write only one declaration per line.
- Indent continuation lines with one tab stop (four spaces).
- Use parentheses to make clauses in an expression apparent.
- Add at least one blank line between method definitions and property definitions.
- Do use vertically aligned curly brackets.

Commenting

- Place the comment on a separate line, not at the end of a line of code.
- Begin comment text with an uppercase letter.
- End comment text with a point.
- Insert one space between the comment delimiter (//) and the comment text.

Documentation

- Use XML documentation comments instead of "simple/traditional" comments to create a proper documentation of your code.
- The "traditional" comments are good to do some clarifications, but they should be used to explain some parts of the code if necessary so whoever checks it, can easily understand how it works.

General

- Do not use underscores except for variables that shouldn't be modified directly. For example: variables used to avoid infinite loops in property setters.
- Avoid using abbreviations except if they are commonly used as names, such as Id, Xml, Ftp, Uri, UI, ...
- Do not use Screaming Caps (all capital).
- Do use predefined type (int, string, bool) names instead of system type names like Int16, Single, UInt64, String, Boolean, ...
- Do name source files according to their main classes. Exception: file names with partial classes reflect their source or purpose, e.g. designer, generated, etc.
- Organize namespaces with a clearly defined structure. Examples: "Company.Product.Module.SubModule", "Product.Module.Component", ...
- When using PascalCasing or camelCasing for words/abbreviations of 1 or 2 chars, they should all be uppercase. Ex: UIControl, FtpTransfer.
- Do not create names which are not clearly differentiated (avoid having the name "example" and "Example").

- Use singular or plural depending on the quantity of elements you are working with.
Ex: Player (singular, a single player), PlayersManager (plural, manages a group of players), Locations (plural, a group of locations), GetNearestPlayer (singular, obtains a one player), GetPlayersInArea (plural, returns multiple players), ...
- Use the "this" keyword only if:
 - There is an ambiguity. Ex: this.name = name.
 - You want to pass a reference to the current instance.
 - You want to call an extension method on the current instance

Strings

- Try to use string interpolation to concatenate short strings.
- To append strings in loops, especially when you are working with large amounts of text, use a StringBuilder object alongside its Append method.

Variables and fields

- Do not rely on the variable name to specify the type of the variable (Hungarian notation⁸ should not be used).
- Use camelCase (even if they have properties/accessors).
- Make all of them private by default.
- Don't show any variable through the inspector if it is not necessary.
- If a variable has to be public use Auto-Implemented Properties⁹ for its definition to limit as much as possible the interaction with it¹⁰.
- Don't modify the accessibility levels for a variable just to change its visibility through the inspector.
 - Instead of making a variable public only so you can see it in the inspector, use the SerializeField modifier.
 - Instead of making a variable private only to hide it in the inspector, use the NonSerialized modifier or the HideInInspector modifier.
- Use meaningful names that clearly state what the variable stores.
- Declare all member variables, fields and attributes at the top of a class, with static variables at the very top.

Statics

- Call static members by using the class name. Ex: ClassName.StaticMember

⁸ Hungarian notation: an identifier naming convention that adds a prefix to the identifier name to indicate the functional type of the identifier (Simonyi, 2006).

⁹ Information regarding the auto-implemented properties can be found in Microsoft's C# documentation through the following link:
<https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/auto-implemented-properties>

¹⁰ Note: If the variable has to be visible through the inspector you'll not be able to use Auto-Implemented Properties, so don't use it in that case.

Singleton

- Use "instance" as name for the field that refers to the singleton instance.
- Usually, if the class is a MonoBehaviour, you might want to initialize the singleton field in the "Awake" method or, if it is not, using auto-implemented properties to get access to it whenever is needed.
- Always check if the singleton field is null before setting a value in it. If it is not, handle the error.

Operators

- Use "&&" instead of "&" and "||" instead of "|".

Object creation

- Create object constructors to be sure that all mandatory variables are set when an object of that class is instantiated.

Class

- Use PascalCasing for the name. Ex: NameOfClass
- Use noun or noun phrases to name a class (Employee, SpawnLocation, Document, ...).

Interface and abstract classes

- Interface names are a noun (phrases) or adjectives.
- Do prefix interfaces with the letter I. (IShape, IShapeCollection, IGroupable, ...).

Methods

- Use PascalCasing for the name. Ex: NameOfMethod
- Use meaningful names for the method that clearly explain the result of the execution of the method. Ex: GetCurrentPlayer().

Arguments and local variables

- camelCasing for method arguments and local variables. Ex: itemCount

Events

- Use "Action<>" instead of creating a "delegate" and then attaching it to an "event" to simplify the code.
- To name the events, add "On" at the beginning of the event's name. Example: OnAdLoaded, OnTimerUp.

- If a method is going to be dedicated just to handle an event, compose his name with the name of the event and add at the end the "Handler" word. Example: OnAdLoadedHandler, OnTimerUpHandler.

Enums

- Use PascalCase.
- Do use singular names for enums with an exception for the bit field enums.
- Do not explicitly specify a type or values of enums (except bit fields).
- Do not use an "Enum", "Flag" or "Flags" suffixes in enum type names.

Good Software Design Practices

Sommerville (2004, pp. 12–13) identified a set of generalized attributes not concerned with what a program does, but how well it does it: maintainability, dependability, efficiency and usability.

Having decided those as the main goals of the objective of applying good software design practices, the design of the software must pursue their inclusion.

The inclusion of **maintainability** can be approximated understanding maintainability as the capacity to provide software maintenance, which has been defined by the International Organization for Standardization [ISO] (2006, p.4) as the totality of activities required to provide cost-effective support to a software system (those activities are performed during the pre-delivery stage as well as the post-delivery stage).

According to the International Electrotechnical Commission (2015), "**Dependability** includes availability, reliability, recoverability, maintainability, and maintenance support performance, and, in some cases, other characteristics such as durability, safety and security". So, seems logic to assume that incrementing the presence of those aspects should turn into an increase in the level of dependability.

To increment the **efficiency**, the objective should be to increase "the quality of doing something well and effectively, without wasting time, money, or energy" (Longman Dictionary of Contemporary English Online, n.d.)).

Usability is not only needed to achieve the best design practices in the final product, but it is by itself one of the secondary objectives. So, if an increase in **usability** is required, the product should be able to be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use (International Organization for Standardization [ISO], 2018, sec. 3.1.1).

Finally, it might be worth mentioning one of the practices that could help achieve the wanted attributes: **modularity**. If applied, the components of the asset should be as less coupled as possible, so one feature does not rely on any other. This way, the users of the asset can remove the tools that they do not need without any trouble. In addition, making the asset modular would allow it to be upgradable, updatable and customizable much easier.

SCHEDULE AND METHODOLOGY

The first issue to be addressed will be finding the errors to fix in the existing asset.

After it, a study of the most important features that could be added to Unity will have to be done.

To figure out which are the most important features missing in Unity multiple approaches can be used such as:

- Personal notes taken during several projects made using Unity.
- Reading the Unity Forums for anything tagged as "Feedback" or "Feature Request".
- Making surveys that could be shared in social networks with active Unity users such as Twitter, Reddit, Unity Forums, Discord servers and messaging apps.

The next objective will be deciding the best metrics to evaluate the value of each possible feature to integrate.

Once the evaluation metrics are decided, a rating of each feature will be done to find its value.

After rating the features, they will be implemented in order from most valuable to least valuable until there is no time left to add any of the new features or all of them are already implemented.

In each new feature or error fix, these steps will be followed to implement them:

1. Describing the exact needs and objectives of it.
2. Investigating to try to find the most optimal way to implement it.
3. Design the different elements that will compose it and its relations.
4. Building it.
5. Creating a test that would check its proper behaviour¹¹.
6. Transforming the test into a demo to serve as an example.
7. Updating the asset documentation to include information about the new feature.

During all the process, each step will be documented and, at any point, it is possible to go back any number of steps to improve what has been already done or address issues found later in the process.

At least once per month, an update of the asset including all the new changes will be published in the Unity Asset Store.

¹¹ The creation of a test or a demo might not be always needed if already exist a way to test it or demonstrate it.

After late May, the work on new features will stop and the latest update of the asset will be published into the Unity Asset Store.

Afterwards, a conclusion of the thesis will be created including a critique to the development process and the feature selection in addition to a review of the performance of the asset in the Unity Asset Store including downloads, reviews and other metrics available.

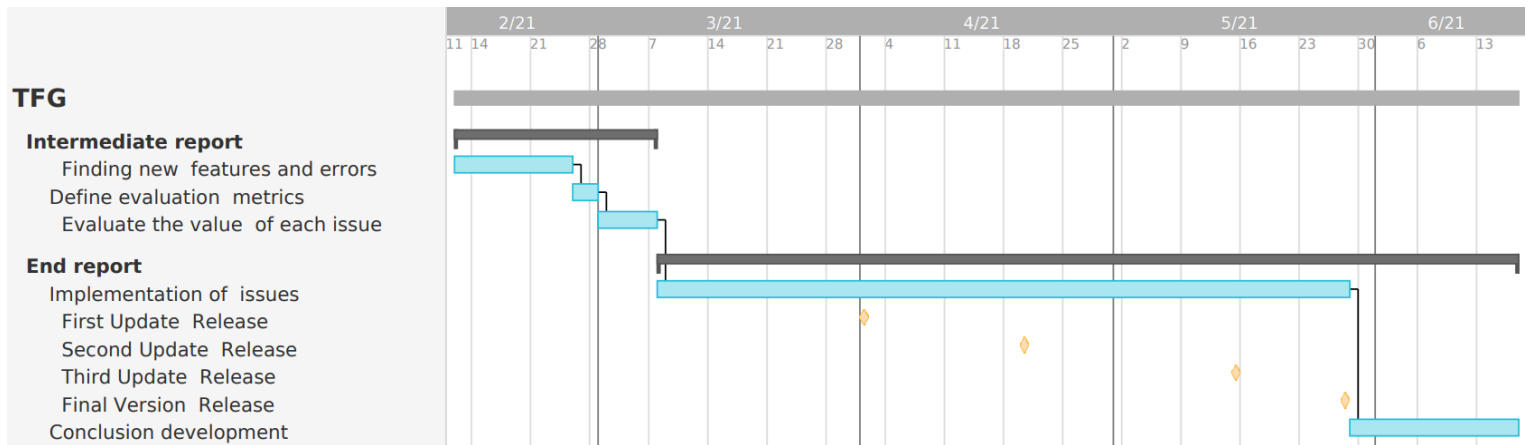


Figure 4. Gantt chart of the schedule for the different tasks and milestones of the project.

PROJECT DEVELOPMENT

1. Finding weaknesses in the existing asset

Comparing the current version¹² of the "Essentials" asset with the Unity guidelines found, good software design and methodology practices and the project objectives, it has been detected that some features are not properly aligned with the theory.

All the elements that have been found that could be modified to match the project's requirements are:

- The changes in configuration for the project have to be manually requested and looked for. This ends up meaning that most of them are unknown for most of the users who do not read the documentation.

Instead, it would be nice to have a window popping up right away after installing the asset asking what configuration do you want to set up for the project. In that window it would be interesting to have the following options:

- Installing the QuickSettings package.
 - Disabling the Warning C60649.
- To debug an enumerable you have to use the class "DebugPro" instead of the default class named "Debug", which would be expected.
- The "Pool" class should have some behaviours modified, functionality added and interaction methods improved in order to mimic as much as possible the usual way to interact with a native Unity class or component. Those modifications are:
 - The "Pool" class should be serializable/editable through the inspector.
 - You should be able to set the parent of the spawned object with the "Spawn" method.
 - The "Pool" class should be able to instantiate a defined number of objects per frame at its creation or when decided to increase the performance.
 - The "Pool" class should be able to pool multiple objects at the same time deciding if it cycles through them in order or randomly.
- The component "SimpleAnimationManager" should be renamed to "ComponentAnimationManager". It better fits its purpose, and it is more self-explanatory.

¹² The version of the asset used as the base for this project is: 1.1.13.

- The component "SimpleAnimationManager" should have a clearer inspector interface to increase the usability and ease of use.
- The class "SimpleAnimation" should be renamed to "ComponentAnimation". It better fits its intent, and it is more self-explanatory.
- The class "EasyRandom" should be renamed, so it starts with the word "Random", so it is easier and more intuitive to find, so the usability is increased.
- The shortcut to clean the console should be displayed while hovering on top of the "clean" button of the console window or in the dropdown options of the console window itself, not in the Essentials file menu entry.
- The Essentials file menu entry should be found in an entry called "Tools", not in the root.

It is possible, however, that additional possible improvements might be found during the development process of the new features or the fixing of the already detected weak points.

2. Finding the most common needs

As mentioned in the "Schedule and Methodology" section, in order to find as many features missing in Unity as possible, multiple techniques have been used:

- The taking of personal notes during several projects made using Unity.
- The reading of Unity Forums posts tagged as "Feedback" or "Feature Request".
- The sharing of surveys in social networks with active Unity users such as Twitter, Reddit, Unity Forums, Discord servers and messaging apps.

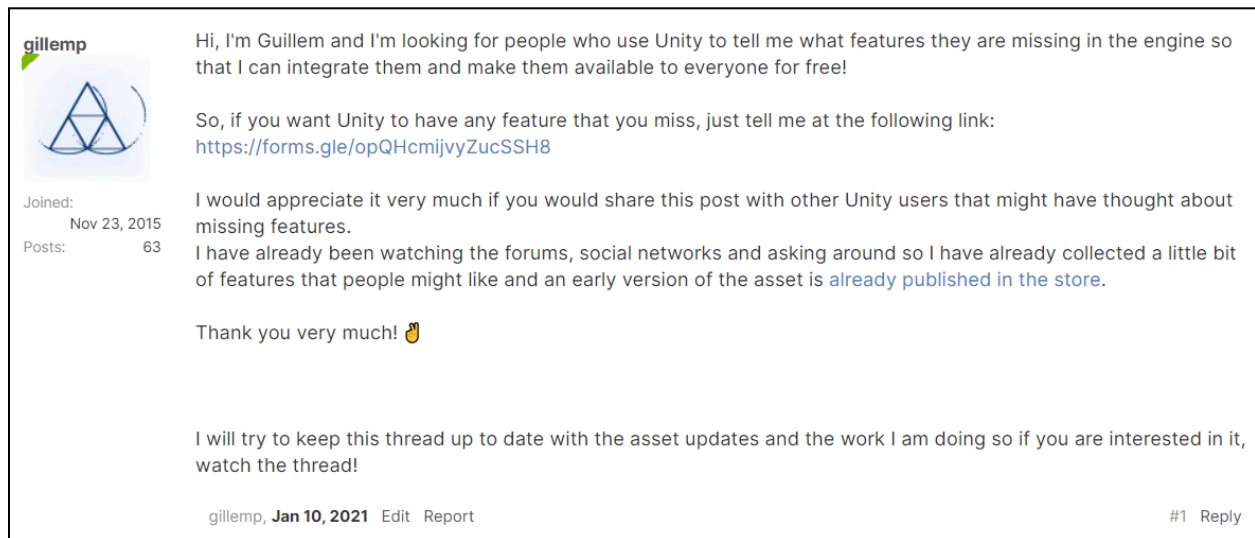


Figure 5. Post made in the Unity Forum to ask the users of Unity which missing features they would like to have integrated in the engine.

All the features that have been found missing using those methods are the following:

- A default template state machine.
- A way to animate Components (camera, transform, ...) without needing an animator controller.
- A way to snap an object to the surface of any other's mesh without leaving any distance between their meshes.
- Ability to create folders in the Hierarchy.
- Ability to encrypt PlayerPrefs.
- Ability to merge lightmaps in additive scenes.
- After the creation of a new project, display a pop-up asking if is wanted to enable the automatic generation of lighting data.
- An alignment system to snap objects into grids (support for multiple types of grids such as square, isometric, triangular, ... would be appreciated).
- An easy way to set the default import settings for different types of assets.
- An integrated .gitignore in the Unity projects or a way to create it natively.

- An orientative folder structure automatically created for the project.
- Asking for the shortcut to trigger the "quick search" after installing the package.
- Audio manager component able to easily handle multiple audio clips.
- Being able to copy and paste the values of a Transform Component world cords even if the object has a parent object.
- Being able to display the console in the game window/screen (an in-game console).
- Being able to set objects as not editable in the hierarchy of the scene.
- C# classes integrating common needs (such as a "rotator" that could rotate between different outputs every time a new one is requested).
- Default custom renderers.
- Default existing presets for components like the camera, canvas, ...
- Default shaders and particles that could be used directly or as template.
- Easy way to animate ragdolls and inverse kinematics.
- Easy way to save c# objects in a JSON format (in a similar manner that PlayerPrefs work).
- Extensions for components and classes like the Transform or the Vector.
- Improved inspector tools to easily create custom windows and inspectors.
- In the game window, alongside the current predefined aspect ratios, display other aspect ratios labelled as commonly used devices.
- In the scene view, being able to select objects behind other objects.
- Integrated networking system.
- Lorem ipsum text generator for the UI components with text fields.
- Straightforward way to download data with HTML requests.
- Templates for character controllers of different types (like first person, third person, ...).
- Visual Scripting system.

However, not all of them are going to be considered to integrate in the asset. A filtering has been made removing those that would transform the objective product from a multi-tool asset to a single-feature asset due the lack of time and how much they would take to be properly created (sometimes because of the amount of work needed to properly incorporate them or the lack of knowledge on the topic).

The features that would get into that category are:

- Ability to merge lightmaps in additive scenes.
- Improved inspector tools to easily create custom windows and inspectors¹³.
- Integrated networking system.
- Visual Scripting system.

¹³ Even though this feature could be at least partially integrated, it has been decided to exclude it since many assets with it as the main objective already exist (such as "Odin - Inspector and Serializer").

3. Deciding the metrics

The payoff and difficulty of each feature has to be calculated in order to prioritize them appropriately.

To keep the project aligned with its main objective, "time" must be directly related to the rating of each feature's payoff. The payoff value must be bigger as more time each feature saves from the average Unity user.

The implementation difficulty of each feature can be also related to "time". However, this time, it will be related to how much time it takes to implement it.

Payoff

To calculate the payoff value of each of the noted features, a poll has been made asking Unity users about how many projects they carry on Unity, the working sessions and hours that they spend using the software¹⁴.

Having the usage metrics about Unity, an approximation can be done in order to guess how much the average user would use each of the new features and, therefore, calculate the payoff of each one of them. Some information obtained through the poll suggested that the median of projects created every year is 3 (0.25 per month), the median of the monthly working sessions are 5 and the median of the monthly hours using the software is 22.5 (4 hours per session according to the median).

Knowing that information, some assumptions can be done regarding how many times per month a tool or feature would be used depending on when it would be employed:

- **Once per project:** 0.25 times per month.
- **Once in every working session:** 5 times per month.
- **Every hour:** 22.5 times per month.

The amount of time per month a feature is expected to be utilized has been used as the payoff of itself. To calculate it, 3 metrics have been used:

- **Average monthly usage:** How many times a month the feature could replace the work of a user trying to work around the missing feature or by trying to create the feature by itself (if feasible).
- **Expected workload reduction:** Every time the feature is used, how much time has been saved from trying to work around the lack of its existence or creating it?
- **Rate of projects where it is usable:** The percentage of projects that this feature could be useful for.

¹⁴ The detailed results of the poll can be consulted in the following document:
<https://drive.google.com/file/d/1yUUC7OFYoAk5rk5e0w4GvJEhD9MCnPdH/view?usp=sharing>

Multiplying all the values, the payoff is obtained, and the only remaining thing to be able to use a PICK chart to prioritize the features is the investment needed to develop and integrate it in the asset. To get it, the personal experience and knowledge on the topics has been used to try to get the most realistic approximation.

Difficulty

To calculate the difficulty of the implementation of each feature, as explained earlier in the theoretical framework, the time of development and the gathering of the information needed is going to be the main point.

So, wanting to approximate as good as possible the amount of time needed to implement each one of them, personal experiences from past Unity projects, personal knowledge about the needed topics and personal time-availability is going to be used.

4. Rating the features

It is important to remark that all the values for each metric have been decided taking into account the own experience, the known data regarding the usage of Unity, the personal knowledge of the software and the personal availability to work on the project.

After doing so, these were the considered possible features to integrate that the people most frequently requested with their respective analysis:

| Formal Name | Expected monthly usage | Expected workload reduction (hours) | Expected rate of projects where it is usable | Calculated Payoff | Expected investment (time to develop) | Calculated Value |
|--|------------------------|-------------------------------------|--|-------------------|---------------------------------------|------------------|
| A way to animate Components (camera, transform, ...) without needing an animator controller. | 1 | 2 | 90% | 1.8 | 5 | 0.36 |
| C# classes integrating common needs (such as a "rotator" that could rotate between different outputs every time a new one is requested). | 0.75 | 1 | 75% | 0.6 | 2 | 0.28 |
| A way to snap an object to the surface of any other's mesh without leaving any distance between their meshes. | 45 | 0.1 | 100% | 4.5 | 20 | 0.23 |
| Default existing presets for components like the camera, canvas, ... | 1.5 | 0.5 | 60% | 0.5 | 2 | 0.23 |
| Extensions for components and classes like the Transform or the Vector. | 1 | 2 | 60% | 1.2 | 6 | 0.20 |
| Audio manager component able to easily handle multiple audio clips. | 0.25 | 7 | 90% | 1.6 | 10 | 0.16 |
| Being able to copy and paste the values of a Transform Component world cords even if the object has a parent object. | 22.5 | 0.1 | 100% | 2.3 | 15 | 0.15 |

| | | | | | | |
|--|------|------|---------|-----|----|------|
| Being able to display the console in the game window/screen (an in-game console). | 0.5 | 3 | 75% | 1.1 | 8 | 0.14 |
| An easy way to set the default import settings for different types of assets. | 1 | 5 | 80% | 4.0 | 30 | 0.13 |
| Being able to set objects as not editable in the hierarchy of the scene. | 10 | 0.5 | 100% | 5.0 | 40 | 0.13 |
| Easy way to save c# objects in a JSON format (in a similar manner that PlayerPrefs work). | 0.25 | 10 | 75% | 1.9 | 15 | 0.13 |
| A default template state machine. | 0.25 | 10 | 70.00 % | 1.8 | 15 | 0.12 |
| Default shaders and particles that could be used directly or as template. | 1.25 | 2 | 35% | 0.9 | 8 | 0.11 |
| Ability to create folders in the Hierarchy. | 15 | 0.1 | 100% | 1.5 | 15 | 0.10 |
| An orientative folder structure automatically created for the project. | 0.25 | 1 | 50% | 0.1 | 2 | 0.06 |
| In the scene view, being able to select objects behind other objects. | 100 | 0.01 | 100% | 1.0 | 20 | 0.05 |
| Straightforward way to download data with HTML requests. | 0.25 | 4 | 30% | 0.3 | 6 | 0.05 |
| Templates for character controllers of different types (like first person, third person, ...). | 0.25 | 20 | 30% | 1.5 | 35 | 0.04 |
| Default custom renderers. | 0.75 | 2 | 25% | 0.4 | 10 | 0.04 |
| Easy way to animate ragdolls and inverse kinematics. | 0.75 | 20 | 10% | 1.5 | 40 | 0.04 |
| Ability to encrypt PlayerPrefs. | 0.25 | 4 | 30% | 0.3 | 10 | 0.03 |
| An integrated .gitignore in the Unity projects or a way to create it natively. | 0.25 | 0.25 | 90% | 0.1 | 2 | 0.03 |
| Lorem ipsum text generator for the UI components with text fields. | 1.5 | 0.1 | 70% | 0.1 | 8 | 0.01 |
| An alignment system to snap objects into grids (support for multiple types of grids such as square, isometric, triangular, ... would be appreciated). | 1 | 0.25 | 40% | 0.1 | 20 | 0.01 |
| Asking for the shortcut to trigger the "quick search" after installing the package. | 0.25 | 0.1 | 100% | 0.0 | 5 | 0.01 |
| In the game window, alongside the current predefined aspect ratios, display other aspect ratios labelled as commonly used devices. | 0.75 | 0.1 | 90% | 0.1 | 15 | 0.00 |
| After the creation of a new project, display a pop-up asking if is wanted to enable the automatic generation of lighting data. | 0.25 | 0.1 | 80% | 0.0 | 5 | 0.00 |

Table 1. Most requested features with it's expected monthly usage, workload reduction in hours, rate of applicable projects, calculated payoff, implementation difficulty (development time) and

calculated value. As greener a cell is, more exceptionally good the feature is, and the reddish a cell is, the worse the feature is.

Knowing the value of each of the selected features, the implementation can be prioritized and the development can start.

5. Feature implementation

The version control through the development process is going to be done using the software "GitHub¹⁵". The project's repository can be found through the following link: <https://github.com/guplem/UnityEssentials>

Fixing the weak points

Automatic application of asset's recommended settings

Description:

The modifications for the project that Essentials provide have to be manually requested and looked for. This ends up meaning that most of them are unknown for some of the users who do not read the documentation.

Objective:

A window should pop up right away after installing the asset asking what configuration the user want to set up for the project. In that window it is wanted to have the following options:

- Installing the QuickSettings package.
- Disabling the Warning C60649.

New options might be added in future updates.

Implemented approach:

An interface and an abstract class have been created to normalize the modifications that essentials easily allows the user to apply. To comply with those normalizations, the existing modification have been updated.

After that, a new editor window named "'Essentials' Settings and Modifications" has been created. In it, all the modifications that inherit from the class "Modification" are going to be displayed alongside short explanations for the available actions.

¹⁵ GitHub is a collaborative development platform to host projects using the Git version control system.

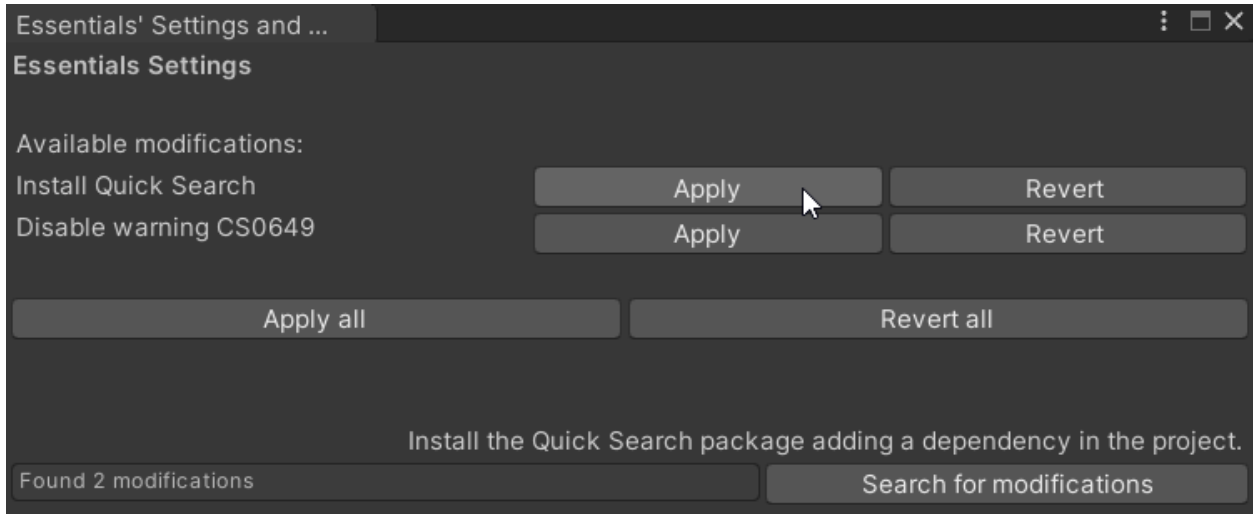


Figure 11. Essentials' Settings and Modifications window with the mouse hovering on the "Apply" button for the "Install Quick Search" modification with its tooltip being displayed.

The "Essentials' Settings and Modifications" window includes a button to search for configuration modifiers not listed on the list in case the automatic population does not work as expected.

Additionally, the window is going to automatically open in the middle of the screen after installing the package in a new project. After doing so, using the "PlayerPrefs" class provided by Unity, that fact is going to be saved in registry, so it does not pop up constantly.

Test and demo:

The testing has been done manually. The window is automatically only just after installing the package, the available modifications are listed and all the buttons perform the actions as expected.

It has been thought that no demo is required to demonstrate this feature because it will be self-demonstrating itself with the automatic pop up.

Result:

The resulting asset after the implementation can be found in the following GitHub link: <https://github.com/guplem/UnityEssentials/releases/tag/Automatic-application-of-asset%E2%80%99s-recommended-settings>

Time of development: approximately 4 hours.

The DebugPro class shouldn't be needed

Description:

To debug an enumerable you have to use the Essential's class named "DebugPro" instead of the default class named "Debug", which would be expected by the Unity users.

Objective:

Integrate the functionalities from the class "DebugPro" into the class "Debug" and delete the need of using the class "DebugPro" or any other class to use the functionality currently existing in the "DebugPro" class.

Implemented approach:

After gathering some documentation it has been concluded that, in c#, you can not use extension methods statically because they are a technique for simulating instance methods.

Other approaches such as creating an additional "Debug" class in another namespace have been considered. But they would end up having the same issue as "DebugPro", you would have to proactively seek for the feature instead of having it available using the default class.

It means that because this class is almost only used statically, the additional features provided by "DebugPro" will have to still be used through this class.

However, an extension method for the features has been created just in case any user tries to use the "Debug" class in a non-static way. And the class "DebugPro" has been renamed to "DebugEssentials" to easily understand from where it comes and its purpose.

Test and demo:

The existing demo scene and script of the feature has been kept in the project and used as testing. No updates were needed because no new specific testing was necessary (apart from the testing on the extension method that has been done in a dummy script).

Result:

The resulting asset after the implementation can be found in the following GitHub link:

<https://github.com/guplem/UnityEssentials/releases/tag/The-DebugPro-class-shouldn't-be-needed>

Time of development: approximately 3 hours.

Improving the Pool class

Description:

The "Pool" class should have some behaviours modified, functionality added and interaction methods improved in order to mimic as much as possible the usual way to interact with a native Unity class or component.

Objective:

Implement the following modifications:

- You should be able to set the parent of the spawned object with the "Spawn" method.
- The "Pool" class should be able to instantiate a defined number of objects per frame at its creation or when decided to increase the performance.
- The "Pool" class should be able to pool multiple objects at the same time deciding if it cycles through them in order or randomly.
- The "Pool" class should be serializable/editable through the inspector.

Implemented approach:

The class was already very compliant with the good software design practices explained before in the theoretical framework, so not many things had to be improved.

To add the possibility to set the parent while spawning a GameObject, an optional parameter has been added to the "Spawn" method that will carry the information about the desired Transform parent.

The automatic instantiation objective has been slightly modified. Before, the objective was to be able to spawn a defined number of enemies per frame. But it has been thought that a more modular approach would be to let the users choose how many objects to load at any given time. So, if they want, they can simply program the way they want their pool to load them. It could be every some seconds, frames or any other trigger.

To implement the possibility to pool multiple objects in the same pool, instead of hosting only one base object, a list of game objects has been added to be used instead. That list will be used by the instantiation process which will choose one them at a time.

Additionally, a random option for the instantiation process of pool has been implemented and added in the constructors. When set to true, the pool is going to pick the objects randomly (not in order) during the instantiation of them. However, after having all the objects instantiated, the respawn of the objects is going to be done in order. It is that way to remove the possibility of respawning one of the latest objects spawned, something usually undesirable.

The serialization process of the class, so it would be editable through the inspector window, only needed to add the attribute "Serializable" to the Pool class in addition to some testing.

Test and demo:

To test the different features, 3 scenes have been created. Each of one of them hosts different ways of using the tool. The scenes have been configured, so they are easier to comprehend and can be used as demos.

- PoolSmall: Demonstration/testing of the simplest way of creating and managing a pool. It is configured through the inspector.
- PoolMultiObjects: It tests and shows that the pool can host more than one object spawning them randomly and how to create the pool through code.
- PoolBig: Tests and demonstrates the loading feature of the tool by loading an object every second. It can be used as a template to load objects every a predefined amount of time. It also shows how the pool can be created through code with just one GameObject as a base instead of a list of them.

Result:

The resulting asset after the implementation can be found in the following GitHub link:
<https://github.com/guplem/UnityEssentials/releases/tag/Improving-the-Pool-class>

Time of development: approximately 4 hours.

Improving SimpleAnimation

Description:

Some aspects of the elements related with the SimpleAnimations are not compliant with the usual way of interacting with Unity.

Objective:

Implement the following modifications:

- The component "SimpleAnimationManager" should be renamed to "ComponentAnimationManager" in addition to the class "SimpleAnimation" which should be renamed to "ComponentAnimation". It better fits its intent, and it is more self-explanatory.
- The component "SimpleAnimationManager" should have a clearer inspector interface to increase the usability and ease of use.

Implemented approach:

It has been chosen not to rename the classes to avoid decreasing the modularity. If they were named so it is understood that they are made to animate components, it would look like their only purpose is to do so, which is not. They can handle the animation process of any class.

The inspector, however, has been improved in multiple aspects. Visual clarity has increased by creating boxes around each animation, not all of them. Additionally, not useful information has been removed. Those elements are:

- The name of the namespace while choosing the animation type (implementation).
- The script name for the manager.
- The editor for the number (size) of the animations.

The implementation's search area has been moved to the bottom and its ascetics have been adjusted to be less attractive due to its lack of constant need of using it. But it has been kept because it can be handy when creation custom implementations of "SimpleAnimation".

Additionally, the text of some labels has been updated to clarify the use of the tool's inspector and a field to name the animation has been added alongside additional functionality related to it. That functionality is the ability to see the animation's name while collapsed in the inspector and being able to play, pause and resume an animation searching it by name in the "AnimationManager".

To finish, a label displaying the type of animation and an animation preview have been added for each animation. In case that the animation affects a Unity Object, it will be marked as dirty, so the state of the object during the preview can be saved.

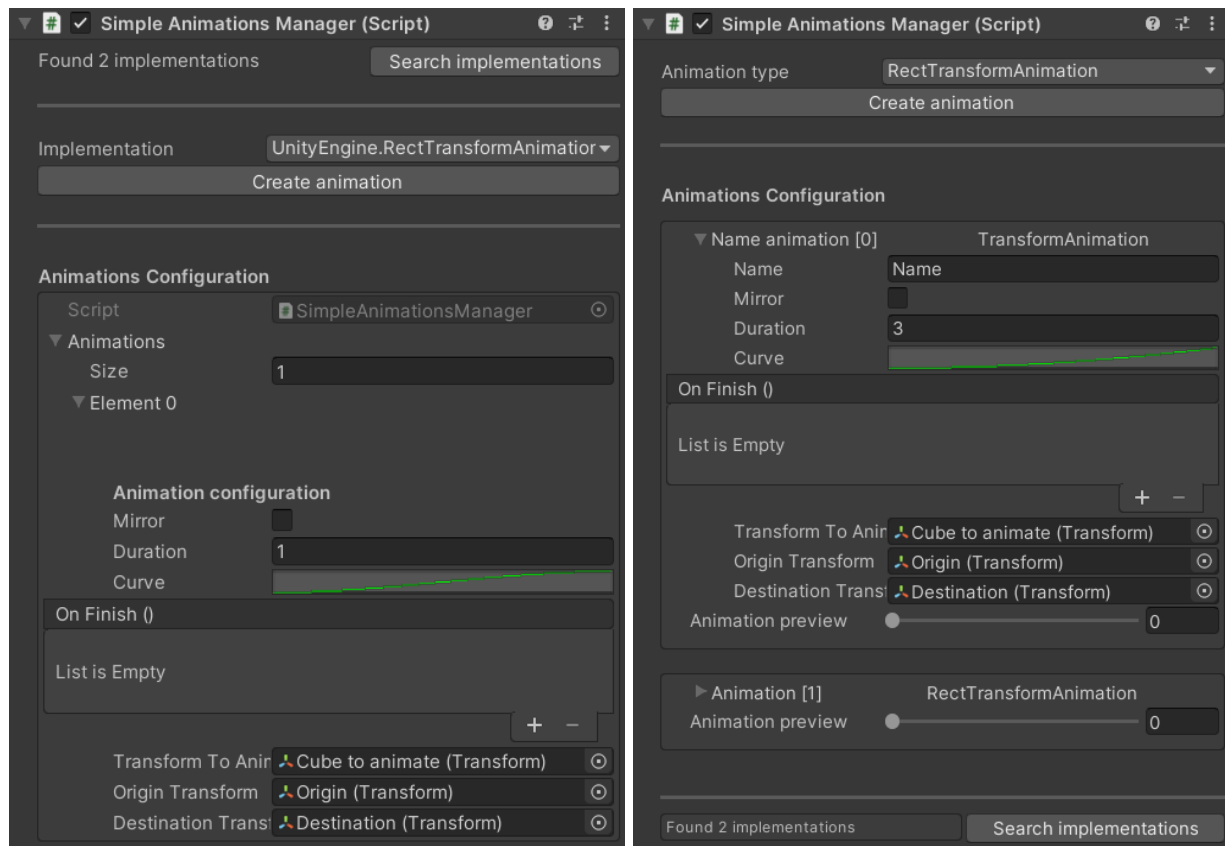


Figure 13. At the left, the old inspector for the Animations Manager. At the right, the new inspector for the Animations Manager.

Test and demo:

The already existing demos have been used as a test environment and kept as demos, so no noticeable modifications have been made.

Result:

The resulting asset after the implementation can be found in the following GitHub link:

<https://github.com/guplem/UnityEssentials/releases/tag/Improving-SimpleAnimation>

Time of development: approximately 5 hours.

Increasing the usability of the EasyRandom class

Description:

The class "EasyRandom" should have a name that starts with "Random", so the usability increases through making it easier to find and therefore, use.

Objective:

The class "EasyRandom" should be renamed, so it starts with the word "Random".

Implemented approach:

The implementation is very straightforward, the only thing needed was to rename the class "EasyRandom".

It has been chosen to name it "RandomEssentials" so it maintains consistency with the "DebugEssentials" class.

Additionally, it has been decided that all future classes that extend the functionality of an existing class and its existence cannot be hid, they will be named as ClasEssentials (where "Class" will be the name of the improved one). This way, they are going to be known by the user easily thanks to the IDE's Automatic Completion features. And, its origin and purpose are more obvious as well.

Test and demo:

The testing has been done in the existing demo scene. No changes were needed.

Result:

The resulting asset after the implementation can be found in the following GitHub link:

<https://github.com/guplem/UnityEssentials/releases/tag/Increasing-the-usability-of-the-EasyRandom-class>

Time of development: approximately 1 hour.

Increasing the usability of the shortcut "Clear Console"

Description:

The "Clear Console" shortcut should not be in the Essentials file menu entry but in a place where Unity users should expect it to be.

Objective:

The shortcut to clean the console should be displayed in at least one of this places:

- While hovering on top of the "clean" button of the console window.
- In the dropdown options of the console window itself.
- In the native shortcuts tool of Unity

Additionally, it should be removed from the Essentials file menu.

Implemented approach:

The "MenuItem" attribute has been replaced by the Shortcut attribute, which allows the user to add shortcuts to the main Shortcut Manager of Unity.

Subsequently, the "MenuItem" for the shortcut disappeared and the shortcut can be seen and customized using the Shortcut Manager.

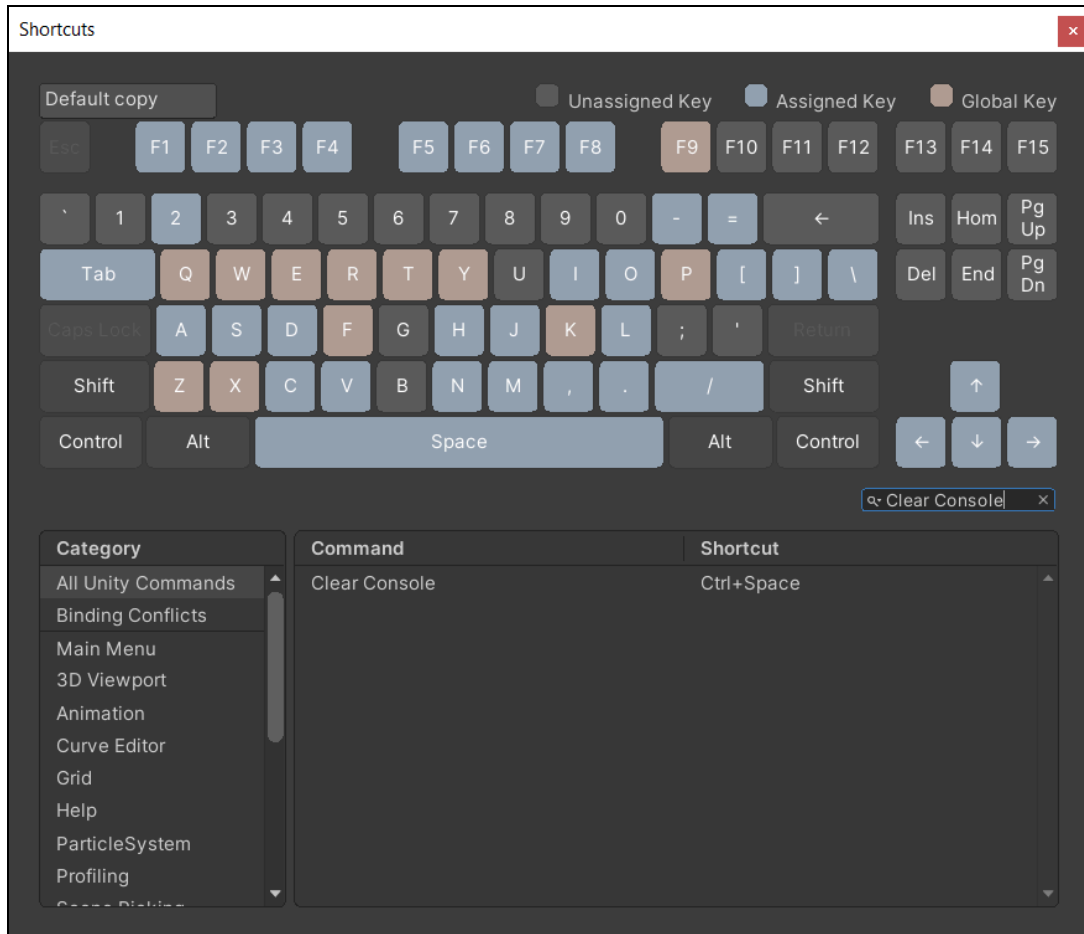


Figure 14. A screenshot of the Shortcut Manager where the “Clear Console” shortcut information it can be seen.

Test and demo:

For the seek of simplicity, it has been decided that small features like shortcuts should not have demos by themselves. They, like this one, will be manually tested and they will appear in the documentation.

Result:

The resulting asset after the implementation can be found in the following GitHub link:

<https://github.com/guplem/UnityEssentials/releases/tag/Increasing-the-usability-of-the-shortcut-%E2%80%9CClear-Console%E2%80%9D>

Time of development: approximately 1 hour.

Improving organization of the Essentials file menu entries

Description:

The Essentials file menu entry should not be found in the root.

Objective:

To have all Essentials file menu entries inside any of the default ones or, if they are not appropriate, under an entry called "Tools".

Implemented approach:

No changes were needed because during the improvements previously made, the desired objective was accidentally recreated before the development of this.

Implementing new features

A way to animate Components (camera, transform, ...) without needing an animator controller

Description:

An easy way to animate more elements commonly used in unity such as components is wanted. This would be used to modify the status of the most common element of the game to a destination state over a defined period of time.

This feature has a high percentage of projects where it could be used while being relatively fast to implement. Given these facts, it got a value of 0.36, the highest of all the rated features.

Objective:

Extending the functionality of the already existing feature under the name of "SimpleAnimation" to add support to animate additional elements such as:

- Camera component
- Vector2
- Vector3
- Colours
- Float
- Integer

Implemented approach:

The already-in-place design of the Simple Animations system allows the creation of different types of animations to handle almost any element that is wanted to be animated in the game.

To add support to animate new Unity elements, the only thing needed is to create a new class that inherits from "SimpleAnimation". The behaviour of the animation has to be programmed in it following the same structure as the already-existing simple animations classes.

Doing so, new classes have been created to add support for all the elements listed as objective with the exception of the "Image" because the only element that could be animated was the colour, which was covered by the "ColorAnimation" class.

Test and demo:

Two scenes have been created to test and demo the newly added features.

- CameraAnimationExample: Displays an example of how the camera can be animated to switch the background colour, field of view, the clipping planes, ratio and location, etc.

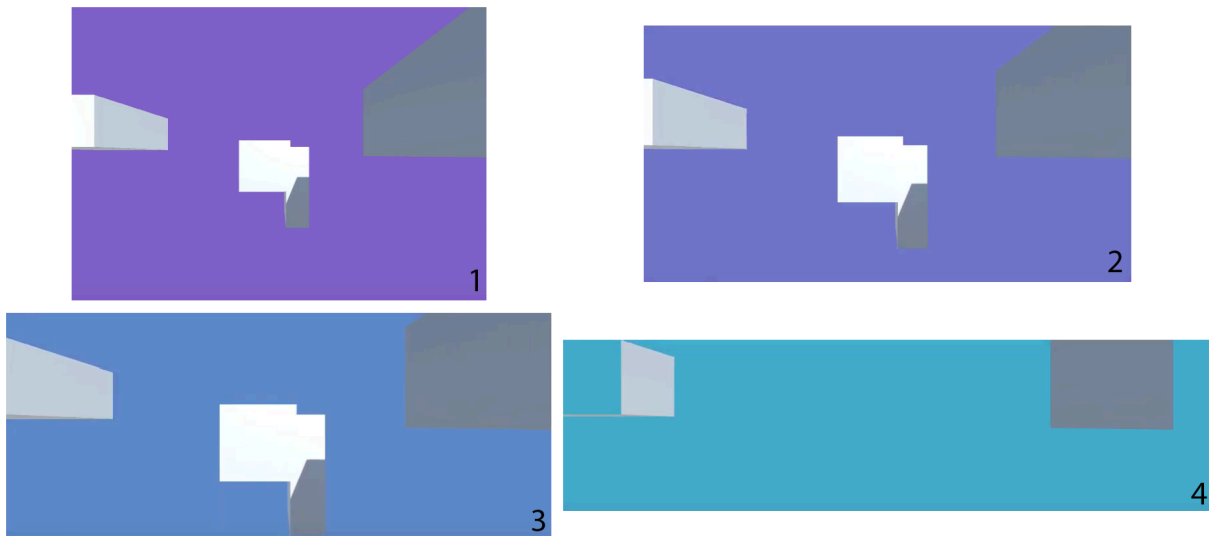


Figure 15. Visualization of the camera animation in four steps (1 to 4). The field of view, background colour, proportions (rect) and clipping planes are animated. Note: Some GameObjects seem to disappear or lose mesh faces, but it is due to the animation of the clipping planes.

- MiscellanyAnimations: Shows how the animation of floats, integers, vectors and colours can be used in the UI or anywhere else.

| | | | | | | | |
|---|-----------------|----------|-----------------|----------|-----------------|----------|-----------------|
| 0 | (0.0, 0.0) | 1 | (0.3, 0.5) | 6 | (1.3, 1.9) | 15 | (2.9, 4.4) |
| 0 | (4.0, 2.0, 0.0) | 19.42384 | (2.9, 3.1, 2.5) | 301.9574 | (0.6, 5.4, 7.6) | 4.419329 | (3.5, 2.5, 1.2) |

Figure 16. From left to right: the animation progressing in the "MiscellanyAnimations" scene, where the colour of an image, an integer, a float, a Vector2 and a Vector3 are animated.

Additionally, a video hosted on YouTube exist as a demonstration of the feature:

<https://youtu.be/tEsQDNpiti>

Result:

The resulting asset after the implementation can be found in the following GitHub link:

[https://github.com/guplem/UnityEssentials/releases/tag/A-way-to-animate-Components-\(camera%2C-transform%2C--\)-without-needing-an-animator-controller](https://github.com/guplem/UnityEssentials/releases/tag/A-way-to-animate-Components-(camera%2C-transform%2C--)-without-needing-an-animator-controller)

Time of development: approximately 5 hours.

C# classes integrating common needs (such as a "rotator" that could rotate between different outputs every time a new one is requested)

Description:

Flow control classes are wanted to mimic the ease of use that they provide in some Game Engines such as Unreal Engine¹⁶.

As the previous features, this one can be used in a high percentage of projects. At the same time, integrating it should take relatively short time. This gives a value of 0.28 to the feature, being the second one from the top.

Objective:

To easily work with the following flow-control nodes/classes:

- DoN (and DoOnce): Allows the call of a function N (or one) time even though they are called more times.
- Sequence (and FlipFlop): Executes one of the linked functions every time it is called rotating between them in sequence or randomly.

Implemented approach:

Four classes (DoN, DoOnce, Sequence and FlipFlop) have been created to allow the users to control the execution flow as desired.

All the classes can be used and configured through the inspector or with code.

Test and demo:

To test and demonstrate the behaviour and usability of the different classes, two scenes have been created. One scene shows how to configure the classes using the Unity's inspector and the other one how to do so through code. In each scene, a game object exist for the demo of each class.

¹⁶ Some examples of nodes to control the flow control can be found in the Unreal Engine documentation:
<https://docs.unrealengine.com/en-US/ProgrammingAndScripting/Blueprints/UserGuide/FlowControl/index.html>

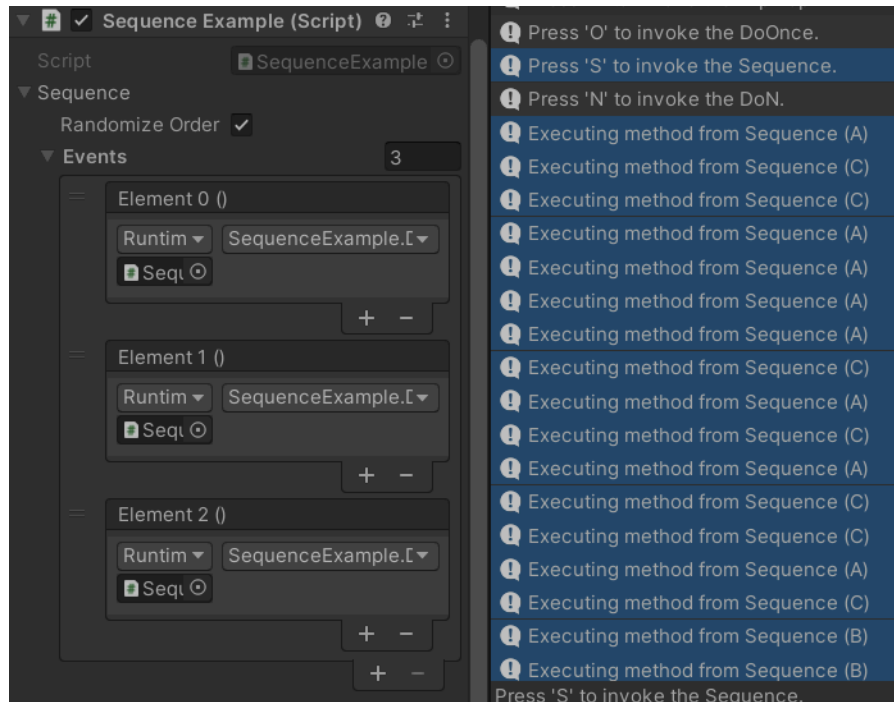


Figure 17. At the left, the configuration view of a "Sequence" in the inspector. At the right, the console displaying debug messages proving the functionality of the "Sequence".

Additionally, a video hosted on YouTube exist as a demonstration of the feature:

<https://youtu.be/KltTSChYFYM>

Result:

The resulting asset after the implementation can be found in the following GitHub link:

[https://github.com/guplem/UnityEssentials/releases/tag/C%23-classes-integrating-com-mon-needs-\(such-as-a-rotator-that-could-rotate-between-different-outputs-every-time-a-new-one-is-requested\)](https://github.com/guplem/UnityEssentials/releases/tag/C%23-classes-integrating-com-mon-needs-(such-as-a-rotator-that-could-rotate-between-different-outputs-every-time-a-new-one-is-requested))

Time of development: approximately 3 hours.

A way to snap an object to the surface of any other's mesh without leaving any distance between their meshes

Description:

Placing objects in a scene can be difficult specially if you don't want any visible gaps between them. To make it easier, a tool to be able to do so is desired.

This feature could be used in almost all projects in almost every session giving it almost the highest payoff of them all. However, the creation and modification of tools for the placement of Game Objects in the scene is, personally, a completely unknown field. So, integrating it might take longer than usual. Overall, the feature gets a total value of 0.23.

Outcome:

During the gathering information about the available tools and ways to implement this feature, it has been found that it already exists as an extension of the "Move Tool" tool for transforms.

To use it, the user must have selected one GameObject in the scene with the "Move Tool" activated at the same time that Shift and Control (Command on Mac) are held. While doing so, a square will appear in the middle of the tool. Dragging that square allows the snapping of the selected object to the surface underneath (at the intersection of any collider).

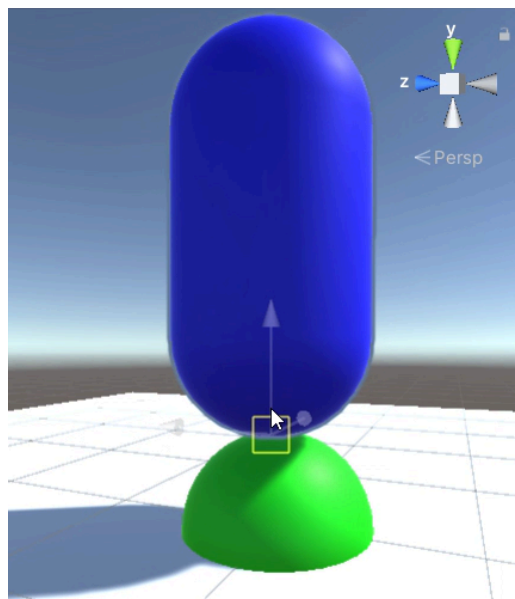


Figure 18. Visualization of the tool in action snapping a capsule on top of a sphere.

More information about positioning GameObjects in a scene can be found in the Unity's Documentation webpage: <https://docs.unity3d.com/Manual/PositioningGameObjects.html>

Additionally, a video hosted on YouTube exist as a demonstration of the feature:
https://youtu.be/V2c_djFiS2k

Default existing presets for components like the camera, canvas, ...

Description:

It is desired to be able to quickly configure components, import assets, ... using prefabs designed to fit as many possible projects as possible. This way, the user could have consistency across projects in addition to ensure that the proper configuration to achieve a goal is easily achievable.

This feature could be used in a quite broad number of projects needing almost no time to implement it in the asset thanks to the ease of creating presets in Unity. This gave this feature a total calculated value of 0.23.

Objective:

During the research phase of this feature, some users gave the feedback that maybe creating default presets might not be that useful because creating your own only takes a few clicks¹⁷.

In addition to this, while thinking about which presets to include, it has been found that, to create the most useful ones in most cases, the number of modifications to make were as few as just one. So, not much workload would be relieved by this feature integration (in contradiction of what was previously thought).

So, it has been decided that, instead of creating presets, the feature will focus on adding a tool to Unity to report if objects or assets related to a selected preset, match it.

The feature will have multiple objectives that will be tried to match in order:

1. A tool to know if all Game Objects of the open scene match a preset (if they contain the component the preset is meant to set).
2. A tool to make all Assets match the presets contained in the same folder.
3. A tool to check if all Game Objects in the open scene match the default presets.

Implemented approach:

To create a tool to know which Game Objects in the open scene have a component not matching the configuration of a preset, a Menu Item has been created. It can be found as an item in the context menu of the presets under the title of "Validate all Game Objects in scene". It will check one by one all the components of the Game Objects in the scene while listing the ones containing a non-matching configuration of the preset component.

¹⁷ Some information about the workflow on creating and using presets can be found here: <https://blogs.unity3d.com/2019/10/11/improve-workflows-validate-decisions-and-avoid-errors-with-presets/>

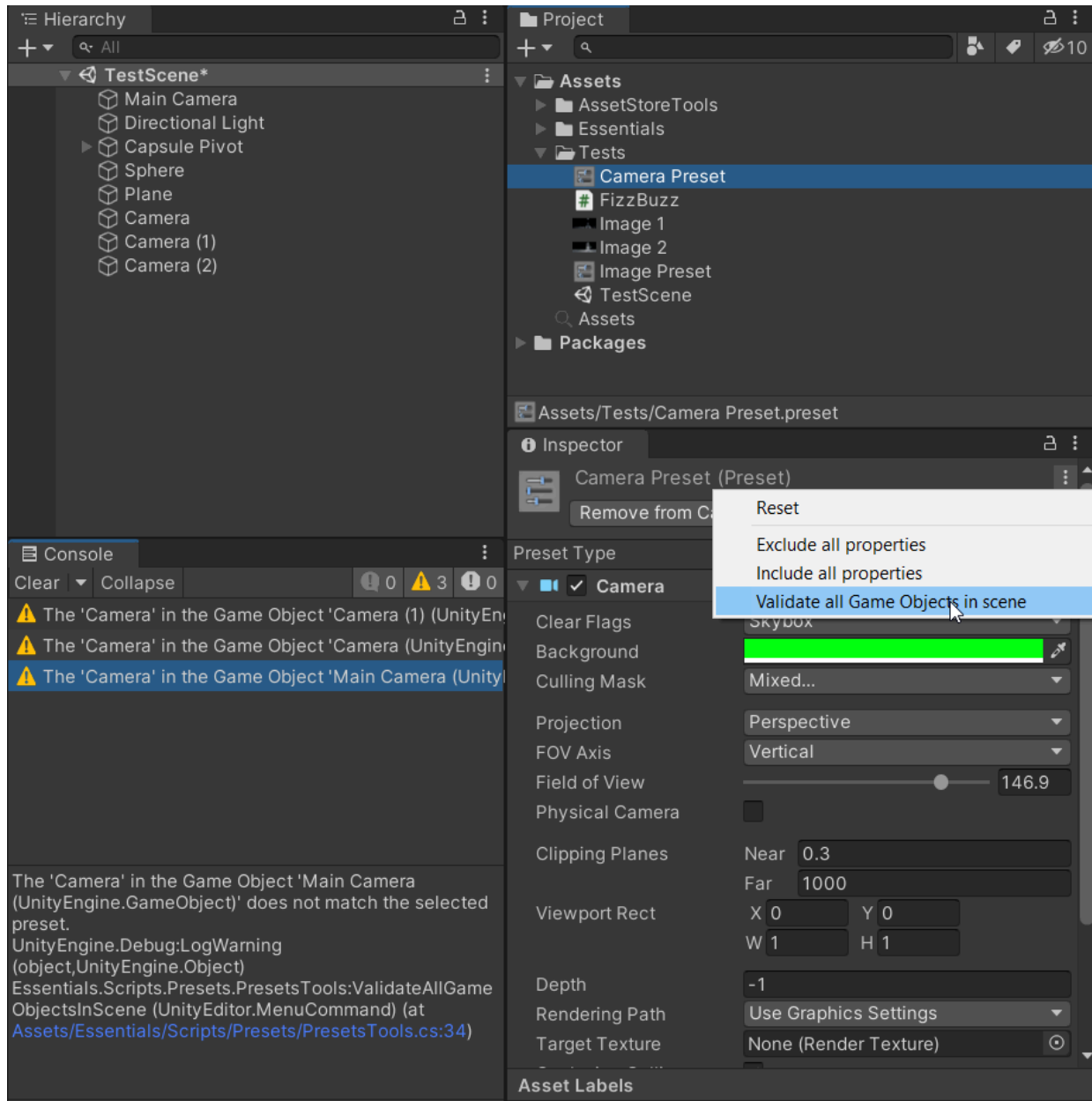


Figure 19. In the inspector it can be seen how the tool is activated while in the console, the results are displayed.

The tool to ensure that all Assets in a folder are configured the same way as the presets contained in it was actually found in a documentation page by Unity found here: <https://docs.unity3d.com/Manual/DefaultPresetsByFolder.html> However, the tool is not integrated in Unity by default even though it has been demanded several times¹⁸ and even acknowledged by Unity. So, the implementation consisted on creating a new "Modification" for the Settings Window of essentials that allows the activation and

¹⁸ Mentions of the feature can be found in this forum thread: <https://forum.unity.com/threads/presets-feature.491263/>

desabilitation of that behaviour by renaming a file so it is (or not) an actual script to the eyes of the Unity engine.

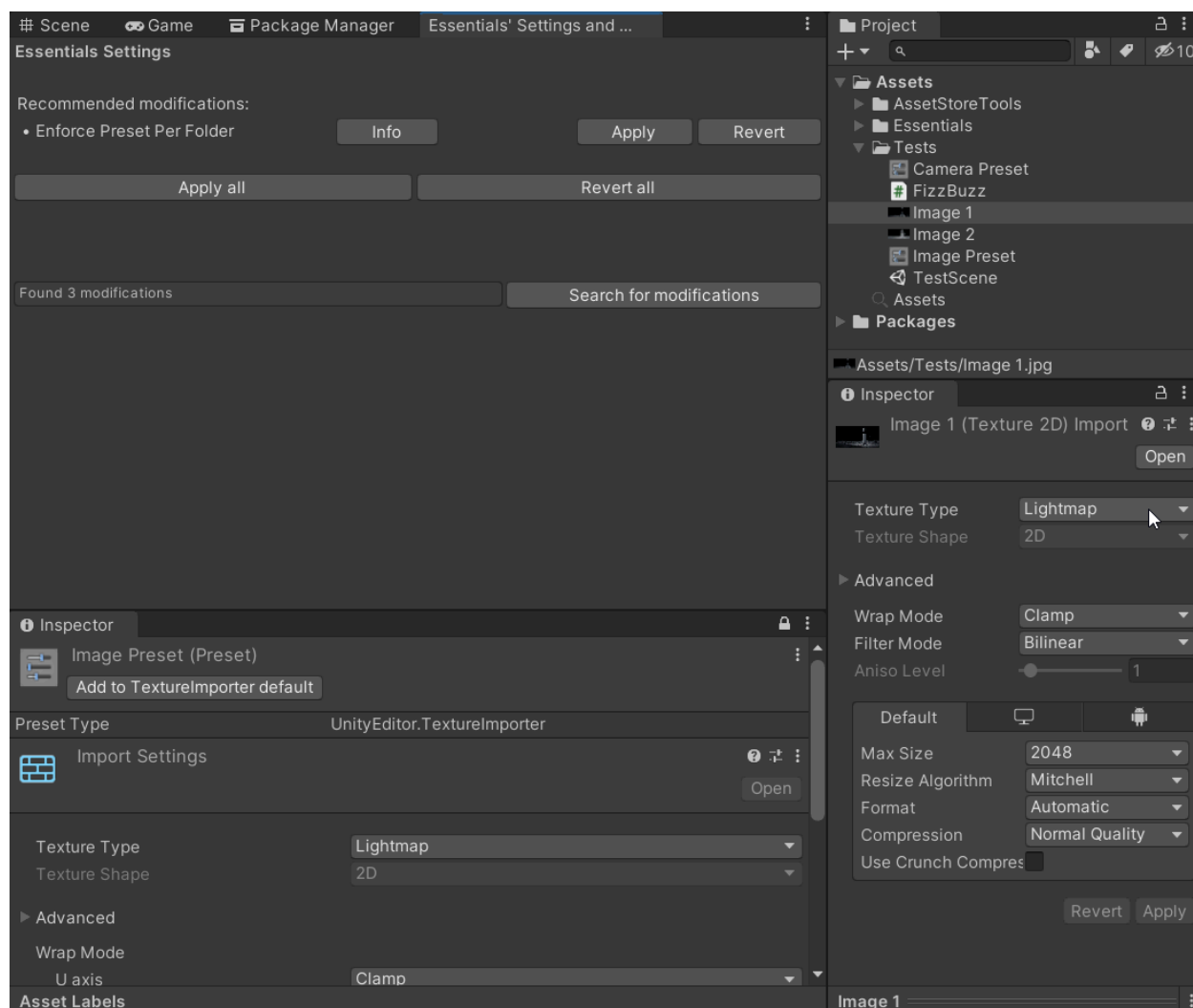


Figure 20. At the top left, the Essentials Settings window showing the modification to activate the tool. At the bottom, two inspectors demonstrate how the preset is applied to the Assets in the same folder.

The last tool: a way to know the Game Objects in the scene match the default presets for their components, was created by adding a Menu Item in the "GameObject" context. This way, it can be accessed simply by right-clicking in the hierarchy of a scene under "Presets/Search mismatches between scene GameObjects and default presets". It behaves like the first tool of this section but, instead of checking all components against one preset, it is checking each component with its defaults.

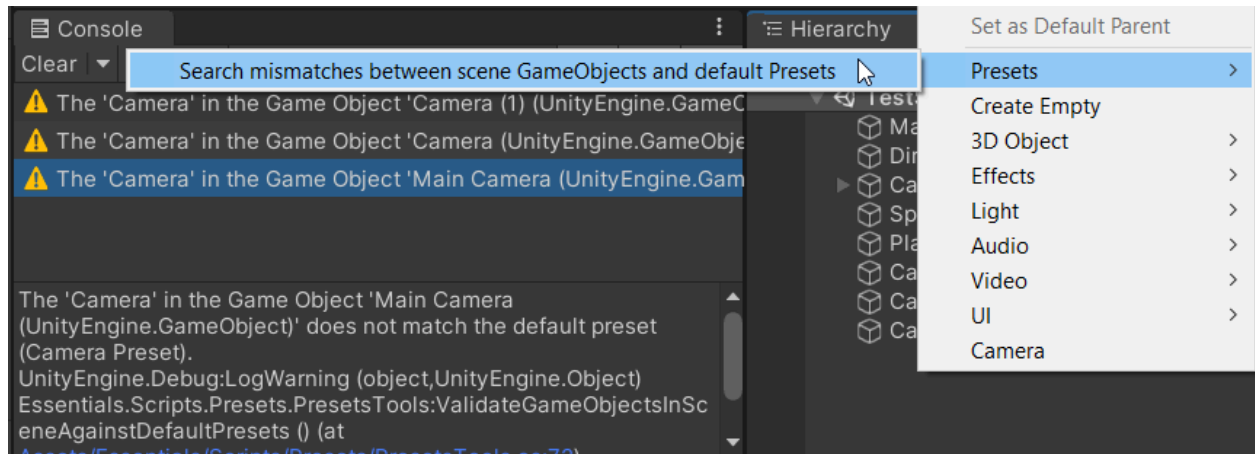


Figure 21. In the hierarchy it can be seen how the tool is activated while in the console, the results are displayed.

Test and demo:

The testing has been done in the development project for Essentials by using a custom scene to check the actual behaviour of the tools against the expected one.

No demo has been created because the use of the tools is very straightforward and one could argue that the most difficult thing about using them is finding them. So, to try to mitigate that, the documentation of the asset has been updated.

Additionally, a video hosted on YouTube exist as a demonstration of the feature:

<https://youtu.be/l-Akd9R3TJs>

Result:

The resulting asset after the implementation can be found in the following GitHub link:

<https://github.com/guplem/UnityEssentials/releases/tag/Improved-Preset-Tools>

Time of development: approximately 6 hours.

Extensions for components and classes like the Transform or the Vector

Description and objective:

An increase of the functionality of the main classes used while using Unity is wanted. The new functionality should allow the users to easily perform commonly needed tasks while working with the engine.

This feature had a right balance between the amount of work needed to implement it and the amount of time it could be used, giving it a value of 0.20.

Implemented approach:

During the development of the original asset and the implementation of the newly added features, sets of extensions have been created or imported to the asset.

Some extensions were actually needed to achieve some functionality for a specific feature, so after properly documenting them, they were left to be used by other users of the asset.

Other extensions were made freely available to use by the authors though the internet. After analysing, improving and documenting them, some extensions not developed by me were added as well.

The resulting extended classes are:

- Camera
- Component
- Debug
- Float
- GameObject
- ICollection
- IEnumerable
- Int
- LayerMask
- Mathf
- Rect
- RectTransform
- SerializedProperty
- String
- Transform
- Vector
- VectorInt

Additional classes might be extended during the implementation of future features.

Test and demo:

The testing has been done in custom scripts and scenes checking the behaviour of the code against the expected one.

No demo has been created for this feature because the code can be used the same way as the native Unity's API and the programming IDEs should help the users use the extensions thanks to the XML comments of each method.

Result:

The resulting asset after the implementation can be found in the following GitHub link:

<https://github.com/guplem/UnityEssentials/releases/tag/Extensions-for-components-and-classes-like-the-Transform-or-the-Vector>

Integrating externally developed features

During the development of the project, multiple improvements of the Unity's engine and workflow made by the community were found and studied.

After a personal evaluation, with the permit of the authors, some of them were added to the Essentials asset either way integrating them directly or enabling its inclusion to the projects through the settings window.

However, an update to the Terms of Service and Package Guidelines of December 4, 2020¹⁹ prohibits the possibility of installing other people's packages not hosted by Unity using another asset or package such as Essentials.

A contact form was sent to the Unity's Support Team asking for an explicit permit to be able to include the feature of installing recommended packages not hosted by unity using the asset Essentials. Having it, the value of the data would increase thanks to being able to use the best tools found and available to reach this project's goal.

However, no such permit was given, and it was clarified that it is no longer allowed to install any asset or package using another asset or package. The only solution would be to have those integrated in the Essentials asset itself or instructing the user how to manually install them.

So, it has been decided that some open and free to use tools might be integrated in the asset. But no intent to create a listing or instruction page for the installation of tools exists for this project due to time restrictions.

¹⁹ Information regarding this update can be found here:
<https://forum.unity.com/threads/updates-to-our-terms-of-service-and-new-package-guidelines.99940/>

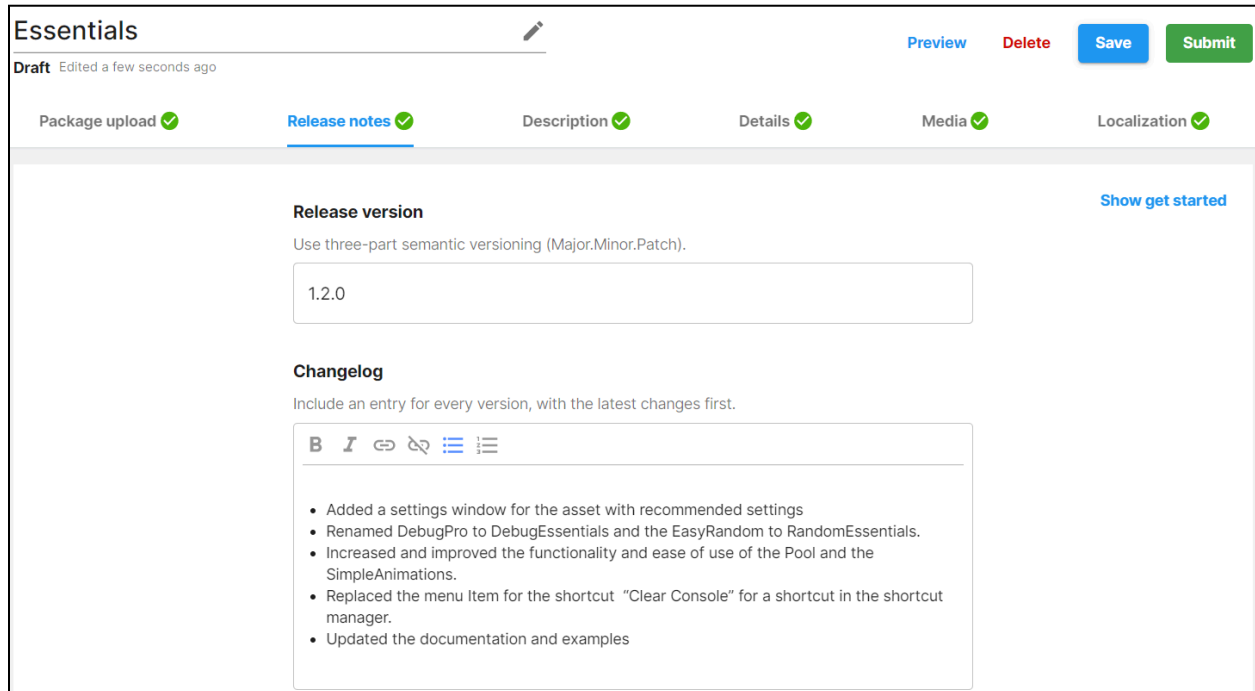
Publishing to the Asset Store

The asset is already available for download since before the start of the project through the following link: <https://assetstore.unity.com/packages/slug/161141>

So, the process of publishing it is not going to be documented²⁰ in this project. However, the process of updating it²¹ will.

To start we have to open the Unity Publisher Portal, which is found in this webpage: <https://publisher.unity.com/packages>. In there, a list of the assets that can be managed can be seen. Clicking on the name of the one that we want to update will open a managing environment for the package.

In there, clicking in “Create a new draft to edit” will lead to a page where it is possible to navigate to the “Release Notes”. In there the changelog and release version can be filled ahead of the update.



Essentials

Draft Edited a few seconds ago

Preview Delete Save Submit

Package upload ✓ Release notes ✓ Description ✓ Details ✓ Media ✓ Localization ✓





Release version [Show get started](#)

Use three-part semantic versioning (Major.Minor.Patch).

1.2.0

Changelog

Include an entry for every version, with the latest changes first.

B I    

- Added a settings window for the asset with recommended settings
- Renamed DebugPro to DebugEssentials and the EasyRandom to RandomEssentials.
- Increased and improved the functionality and ease of use of the Pool and the SimpleAnimations.
- Replaced the menu Item for the shortcut “Clear Console” for a shortcut in the shortcut manager.
- Updated the documentation and examples

Figure 22. View of the “Release Notes” section of the Unity Publisher Portal 2.0 with the information regarding the update to the version 1.2.0

After writing down the information, it must be saved (not submitted). Then, switching to unity is required alongside having the “Asset Store Tools” asset installed for the project.

²⁰ Unity has published a video on YouTube explaining all the documentation process. The video can be found here: <https://youtu.be/Sp7vUE3Hmtw>

²¹ The updating process has been done in the beta version of the “Unity Publisher Portal 2.0”.

Using the Asset Store Tools, the “Package Upload” window can be opened. After logging in with the Unity’s publisher id, the list of the available assets should populate.

Following the instructions in the “Package Upload” window, the next steps are:

1. Select the package to upload from the list (it should have “draft” written at the right because it was created in the previous steps).
2. Select the folder that is containing all the assets relative to the package.
3. Choose if it is needed to include package dependencies, so the asset behaves as expected (in the case of Essentials, no).
4. Validate the package to check that the new version does not have common submission mistakes.
5. Upload the package.

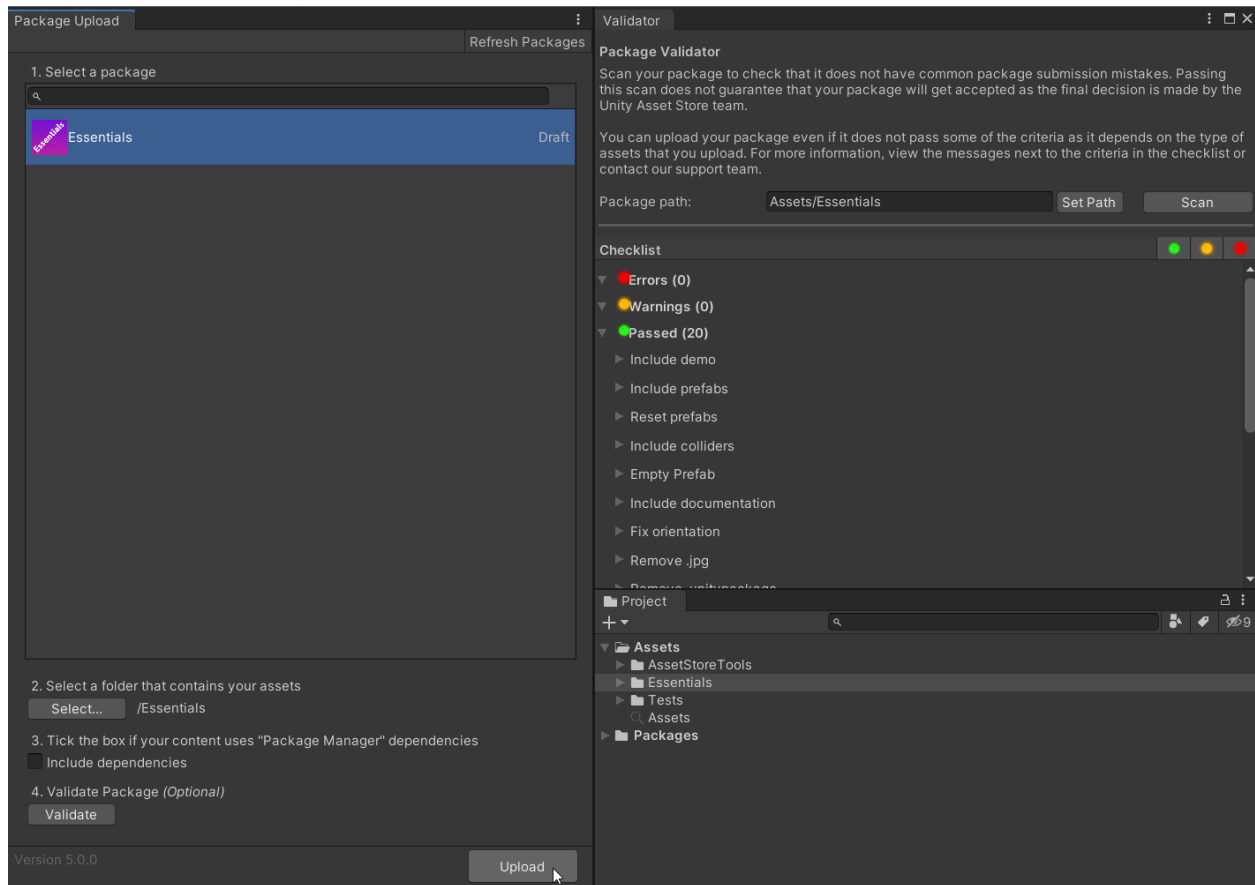


Figure 23. At the left, the Package Upload window. At the top right, the Validator window. At the bottom right, the Project folders structure window.

After uploading the package, checking the “Package Upload” section in the Unity Publisher Portal is recommended to ensure that the upload went through. It can be easily done by checking the time of the last upload.

After ensuring checking for warnings and errors, a submission of the update can be done by clicking on "Submit".

Then, the review process starts. It can take days but, after it, the asset is going to be updated and available for everybody using the Unity Asset Store.

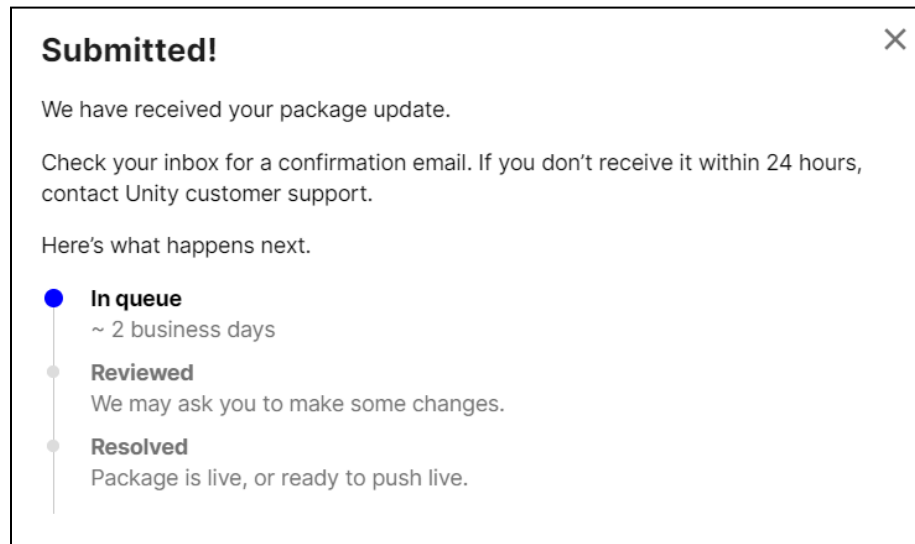


Figure 24. The pop-up displayed after submitting the update for the asset using the Unity Publisher Portal 2.0 informing about the review process schedule.

Changes on each version

Fixing update - 1.2.1 (13th of March 2021):

- Added a settings window for the asset with recommended settings
- Renamed DebugPro to DebugEssentials and the EasyRandom to RandomEssentials.
- Increased and improved the functionality and ease of use of the Pool and the SimpleAnimations.
- Replaced the menu Item for the shortcut "Clear Console" for a shortcut in the shortcut manager.
- Updated the documentation and examples

This release was not programmed in the project schedule, but it has been done in order to validate the updating system and the workflow of the project ahead of time.

GitHub tag link: <https://github.com/guplem/UnityEssentials/releases/tag/1.2.1>

First Update - 1.3.1 (20th of April 2021):

- Added support for Simple Animations of floats, integers, cameras, colours and vectors.

- Added C# classes: FlipFlop, DoOnce, DoN and Sequence to control the flow.
- Added extensions.
- Added tools to increase the functionality of the presets.

GitHub tag link: <https://github.com/guplem/UnityEssentials/releases/tag/1.3.1>

REFERENCES

- Axon, S. (2016, September 27). *Unity at 10: For better—or worse—game development has never been easier*. Unity at 10: For better—or worse—game development has never been easier | Ars Technica. Retrieved January 15, 2021, from <https://arstechnica.com/gaming/2016/09/unity-at-10-for-better-or-worse-game-development-has-never-been-easier/>
- Badiru, A. B., & Thomas, M. U. (2013, April 20). Quantification of the PICK Chart for Process Improvement Decisions. *Journal of Enterprise Transformation*, 3(Quantification of the PICK Chart for Process Improvement Decisions), 1-15.
- George, M. L. (2003). *Lean Six Sigma for Service: How to Use Lean Speed and Six Sigma Quality to Improve Services and Transactions*. McGraw-Hill.
- International Electrotechnical Commission. (2015, February). 192-01-22. IEC 60050 - International Electrotechnical Vocabulary - Details for IEV number 192-01-22: "dependability". Retrieved February 6, 2021, from <http://www.electropedia.org/iev/iev.nsf/display?openform&ievref=192-01-22>
- International Organization for Standardization [ISO]. (2006). *ISO 14764:2006 Software Engineering — Software Life Cycle Processes — Maintenance* (Second ed.).
- International Organization for Standardization [ISO]. (2018). Part 11: Usability: Definitions and concepts. In *ISO 9241-11:2018 Ergonomics of human-system interaction*.
- Longman Dictionary of Contemporary English Online. (n.d.). *efficiency*. efficiency | meaning of efficiency in Longman Dictionary of Contemporary English | LDOCE. Retrieved February 6, 2021, from <https://www.ldoceonline.com/dictionary/efficiency>

Moran, J. W., & Riley, B. (2014, June 27). *PICK Chart*. Retrieved January 30, 2021, from http://www.phf.org/resourcestools/Documents/PICK_Chart.pdf

Schardon, L. (2021, January 2). *Best Game Engines of 2021*. Best Game Engines of 2021. Retrieved January 30, 2021, from <https://gamedevacademy.org/best-game-engines/>

Simonyi, C. (2006, December 07). *Hungarian Notation*. Hungarian Notation | Microsoft Docs. Retrieved January 06, 2021, from [https://docs.microsoft.com/en-us/previous-versions/visualstudio/visual-studio-6.0/aa260976\(v=vs.60\)?redirectedfrom=MSDN](https://docs.microsoft.com/en-us/previous-versions/visualstudio/visual-studio-6.0/aa260976(v=vs.60)?redirectedfrom=MSDN)

Sommerville, I. (2004). *Software Engineering* (7th ed.). Pearson Education.

TXM Lean Solutions. (2020). *Prioritising Improvement Ideas with a PICK Chart*. Improvement Ideas Scoring with a PICK Chart - TXM Lean Solutions. Retrieved January 31, 2021, from <https://txm.com/ranking-improvement-ideas-pick-chart/>

Unity Technologies. (2021). *Unity Platform*. Unity Platform | Unity. Retrieved January 30, 2021, from <https://unity.com/products/unity-platform>