

GENERAL CONSIDERATIONS REGARDING THE DEVELOPMENT OF GAMES USING UNITY TECHNOLOGY

Alexandru TĂBUȘCĂ¹²¹

Cristina COCULESCU¹²²

Mironela PIRNAU¹²³

Abstract:

Video game development platforms represent a specialized work environment that contains a multitude of properties and tools that help game creators in designing and making applications in this field. The market for game development platforms is currently booming, offering a diverse range of options. Anyone with motivation and ideas can start creating and developing a game using popular platforms such as Unreal Engine, Unity, Game Maker or RPG Maker VX Ace. Today, every platform for game development is supported by a vast basic documentation but also by a huge public community that provides game creators with all the tools they need to create high-performance games. If an experienced or even less experienced developer finds himself in a situation where he cannot solve a problem on his own, he has the opportunity to find the answer to the problem encountered, on the community forums corresponding to the technology used in creating the game, or he will find someone who has faced a similar situation and can give him answers/advice for the specific situation. Depending on the purpose of the final product, the game creator has the opportunity to choose the platform he will use in creating it. Unity is the most popular video game development platform due to its cross-platform capability. Unity can be used both in making 2D or 3D games, but also in the film, automotive, architecture, engineering, and construction industries.

Keywords: game development, cross-platform, Unity

1. Introduction

Unity is a cross-platform game engine developed by Unity Technologies, launched first in June 2005, exclusively for MacOS X. It can be used to create games in bidimensional, tri-dimensional, virtual, and augmented reality environments – including simulations [1]. The latest stable version, 2019.2.17, has been launched in December 2019. Even since 2018, Unity has been utilized to develop approximately half of the new mobile games brought to the market and around 60% of the content developed for augmented reality and virtual reality. The programming languages employed in conjunction with Unity are C# and JavaScript. For these two programming languages an integrated development environment is available within Visual Studio of MonoDevelop, as code editors [2].

¹²¹ PhD Associate Professor, Romanian-American University, School of Computer Science for Business Management, tabusca.alexandru@profesor.rau.ro

¹²² PhD Associate Professor, Romanian-American University, School of Computer Science for Business Management, coculescu.cristina@profesor.rau.ro

¹²³ PhD Associate Professor, Titu Maiorescu University, Faculty of Informatics, mironela.pirnau@prof.utm.ro

Unity Technologies, as well as the community members, actively create *assets*. Unity Asset Store is a growing library of such assets, hosting thousands of free and low-price elements that can help save both time and effort for Unity developers. Inside the store there are many types of different assets, including textures, animations, fully-fledged models, complete projects, tutorials and editor extensions. This blend of both free and easily accessible assets (low-cost ones) represents a hugely important feature for Unity, because the developers can download them and insert directly into their project [2]. Today, in pandemic conditions worldwide, for the development of any game the sound component is very important and helps produce a captivating and emotional game experience. Finding the perfect soundtrack that best matches a game application is a very important element to attract and retain players (customers). To this end, Unity offers an extensive library of both free and low-cost audio elements, music and sound effects, so that the game developer should only identify exactly what matches as best as possible to his game. [2][3].

Unity offers a very intuitive user interface that invites developers to experiment with the multitude of options and features offered for content creators. There are three different version of this interface: Personal, Plus and Pro. There also exists an Enterprise version, similar to the Pro one but bundled with an Enterprise environment package.

2. New updates and features in Unity

Unity has brought updates, new features, and tools for the 2D game developers. It added support for maneuvering of animations directly from the “timeline”, for animation preview and modification [2][4]. The application interface is very well organized and very easy to understand and use (see Figure. 1).

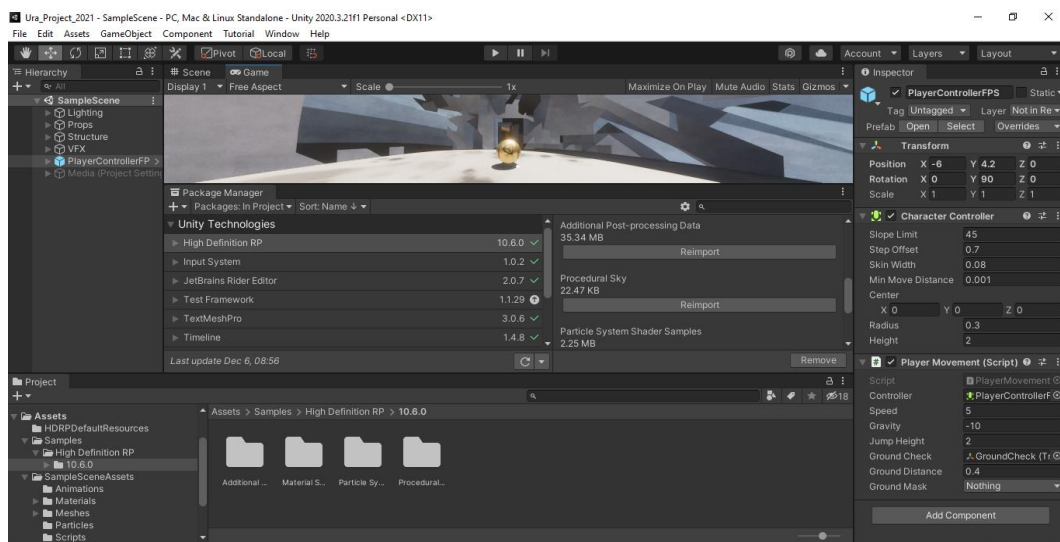


Figure 1. Main Unity user interface

The interface comprises many different options, among which we notice the following main ones: Project (which shows the assets), Hierarchy (through which developers see all elements inside the game environment), Console (shows errors, warning messages or help

error-tracking through the debugger), Asset Store (used for adding diverse objects/assets for the game), Prefab (for different effects for the camera/view, field etc.), the Scene window (where developers can edit all aspects of the current project), and the Play, Pause, Forward buttons. Unity also makes available to the game creators many different scripts that can add functionalities, in the “Tilemap Editor” section, as well as new brushes and tiles which can be accessed from the 2D Extras package [1][5]. The 2D editor for tile maps is not available by default when first installing Unity, but it can be downloaded from the Package Manager section. An important added feature was introduced in Unity as the 2D Sprite Shape component, meant to help create curved fields. The facility goes together with an editor that allows the definition of margins and fillings, as well as the automatic creation of colliders [2]. 2D PSD Importer component permits the import of sprites directly into a third-party application like Photoshop, keeping the image layering – fact that allows layer-by-layer animation design without the need to individually import each individual layer.

MonoBehaviour is the basic class of all scripts meant to be attached to a game object and it is used for faster code creation, lighting manipulation, and development of Artificial Intelligence (AI) modules without extensive previous AI knowledge [2][6].

The main widely used functions of the MonoBehaviour class are:

1. Start() – is represents a function called only once, inside the first frame after script activation, before any Update type function
2. Update() – represents a function called at each individual frame
3. FixedUpdate() – represents a function called regularly for physics calculations (by default every 0.02 seconds)
4. Awake() – a function called only once on all scene objects, before all Start functions, when the script instance is loaded

Useful properties with wide usage can also be mentioned:

1. enabled – when a certain object is marked as active it can be refreshed through the use of an Update function
2. gameObject – it refers to the object to which the component is attached to
3. tag – it refers to the tag associated to the object (it can be added to more objects at the same time, as a category descriptor) and it can be used to compare objects
4. transform – offers access to the position, rotation, and scale of the object and to their manipulation
5. name – it represents the name of a certain object, used for calling it in different scenarios

Important methods also widely used are:

1. GetComponent(Type objectType)
2. SendMessage(string methodName, object value)
3. Destroy(Object object) – it destroys the object

4. DontDestroyOnLoad(Object object) – the object will not be destroyed at the loading of a new scene
5. Find(string name) – it will identify the respective object [2][8].

3. Unity Tools and Technologies

The C# programming language is a general-purpose language, in the category of “strong typing” programming languages, component and object oriented. It was developed in 2000 by Microsoft as part of the .NET project and allows programmers to develop applications that run on top of this system. With the help of C# used within Unity one can define new component, can extend existing components, can define player interactions as well as enemy behavior, can model moving objects or it can actually model the entire game [2][6][7].

Main C# characteristics are:

- Garbage collection – a mechanism that automatically frees the memory occupied by unused objects
- Exceptions management – offers an organized environment for detection of errors
- Focuses on versioning for ensuring computability of different programs during development phases
- Through a Microsoft backed project, called Mono, C# became quite versatile from the point of view of portability. Mono represents an open-source development platform based on .NET, the ECMA standards for C# and the common infrastructure language CLI (Command Line Interface). This fact makes possible the porting of applications on different other platforms, such as Linux-based ones, macOS, Sony PlayStation or Xbox consoles.

Encapsulation, known also as the concept of “data hiding”, is defined by the grouping of the code together with the data it manipulates, thus restricting the access to these data from outside the class. Among the advantaged of the encapsulation method, we mention hiding the class implementation (code) from external users, flexibility and modularization possibilities.

C# code is reusable and easily editable. The inheritance concept, within the object-oriented paradigm, permits the defining of derivative classes from a master class or overloading of its functions. Polymorphism represents the third basic concept of object-oriented programming, and it is defined as the overloading of the master class methods in order to purposely manipulate objects from the derivative class [2][8].

Scripting and its role

Scripts are a key ingredient in implementing any Unity application. A script is built from a list of commands that are executed by a certain program. Those commands are used to

automatize processes. In Unity, C# or JavaScript scripts are attached to scene objects in order to determine their behavior. For adding a script to an object component, one has to use the Add Component button from the inspector section. The Unity engine uses for any implemented script a system of event methods which are predefined (they exist even if not specifically declared inside the script class). Unity permits developers to create their own components by using scripts to trigger game events but also to modify game components' properties [6][8].

A unity script is created and implemented as shown within Figure 2.

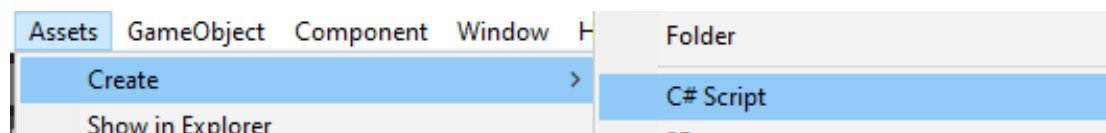


Figure 2. Creation of C# script in Unity

The script is created inside the folder selected from the Project panel. By default, Unity will use Visual Studio for script editing, but one can select any other editor from the Preferences -> External Tools section (see Figure 3).

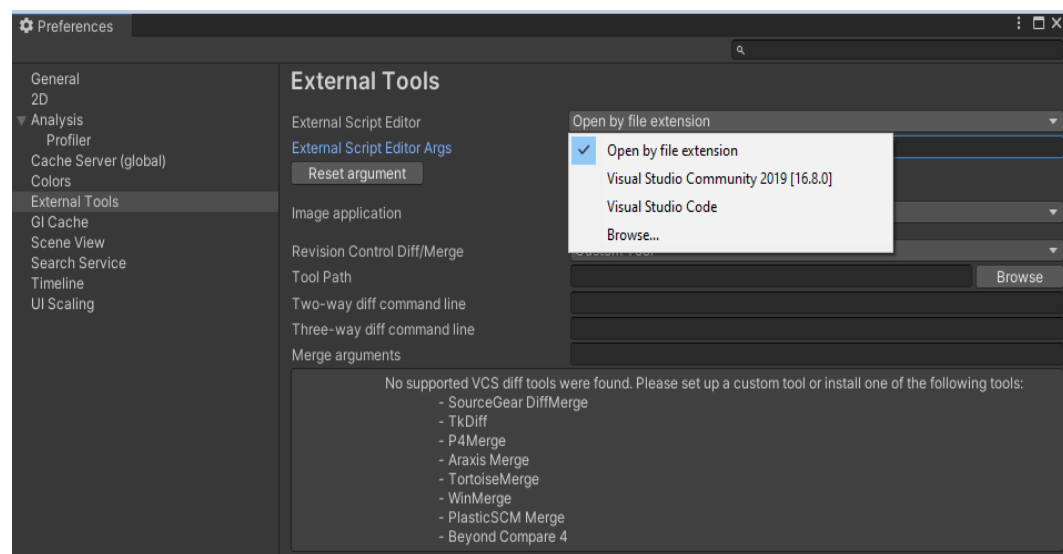


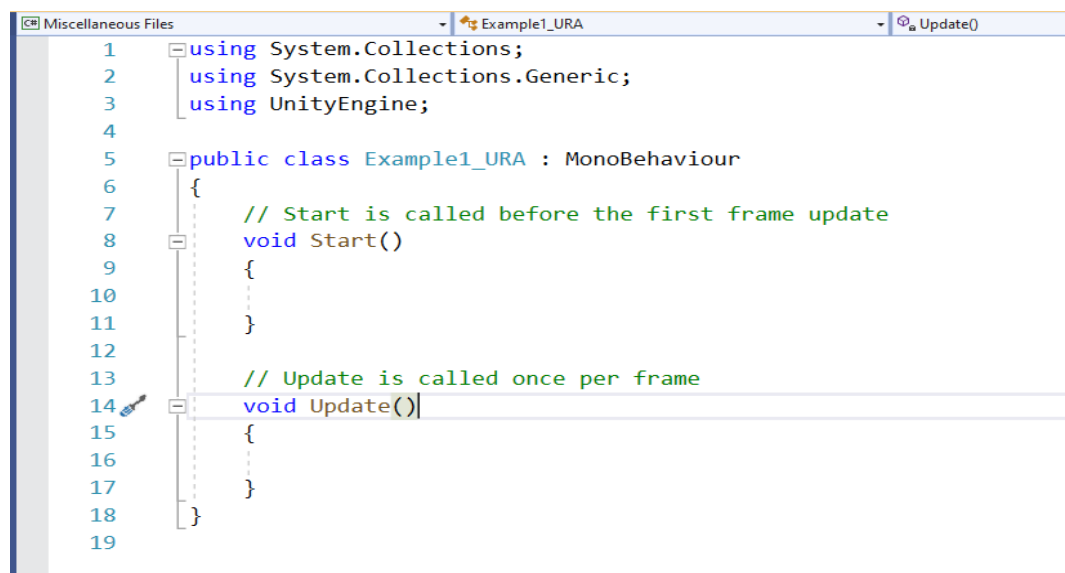
Figure 3. Setting the editor for script creation/editing

The initial content of the script file looks like Figure 4. The script makes the link with the internal Unity engine by implementing a class that derives from the encapsulated Unity native class MonoBehaviour.

If one attaches a script component to a game object, this creates a new instance of the project defined object. The name of the class is taken from the name given when creating the script

file. The name of the class and the name of the file must be the same in order to allow the attachment of the script component to the game object [2] – in our example the name used is “Example1-URA”, as seen inside Figure 4.

The Update() function is the place that hosts the code which will take care of the frame update for the game object. This process includes the movement, triggering of actions and answers to user input. It is also very useful to be able to configure variables, read preferences and create connections with other game objects before any action actually takes place inside the game. The Start() function is used to cover all initializations and will be called by Unity before the first call of the Update() function.



```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class Example1_URA : MonoBehaviour
6 {
7     // Start is called before the first frame update
8     void Start()
9     {
10
11     }
12
13     // Update is called once per frame
14     void Update()
15     {
16
17     }
18 }
19
```

Figure 4. Default content of a Unity script

Inside Unity, initializing of an object is not done by using a constructor function, because the instantiation of the objects is managed by the editor and does not take place at the beginning of the game only. One can attach a script by dragging the script inside a game object, from the hierarchical panel or inside the selected game object inspector. The predefined element acts as a template based on which we can create new instances of that predefined element. A scene is an interactive “window” from Unity to the current game being developed. This contains the hierarchy of the inserted objects, the camera, the lights, the canvas, prefabs etc. Immediately after a scene is created it must be added to the scenes array, accessible from the File -> Build Settings tabs (see Figure 5).

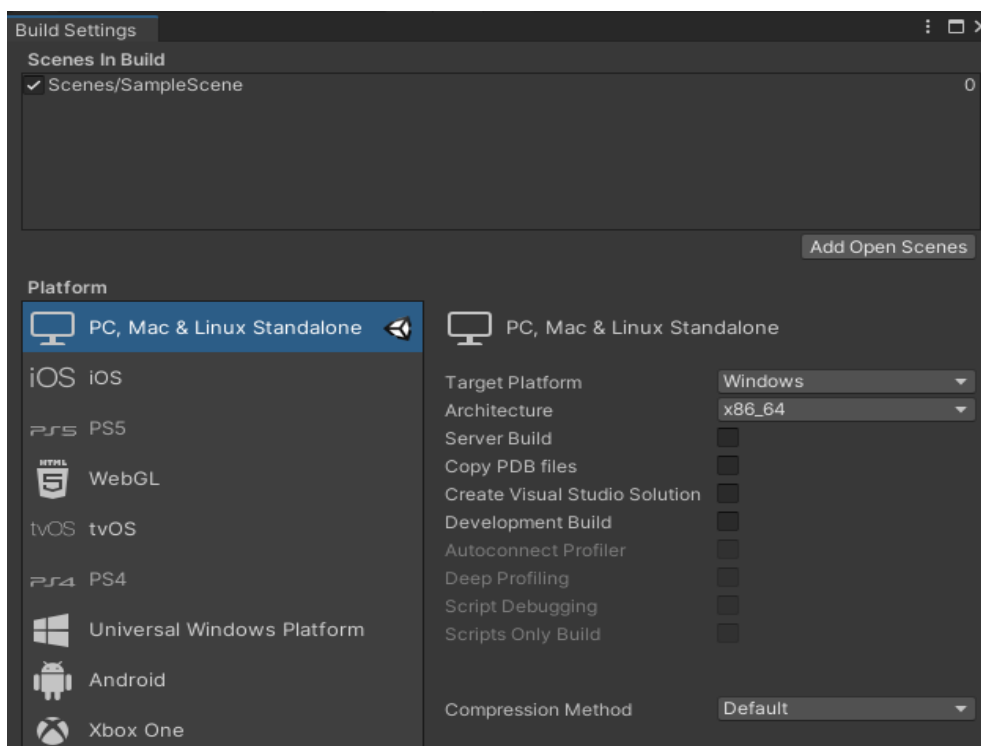


Figure 5. Build Settings tab

The order of the scenes is very important because the passing from one to another can be done only from top to bottom. The prefab is a template type object which permits the storage of an object with all its elements, to be later reused anytime one needs. The prefab can be used as a game object inside scripts, is saved with all its settings, components and added scripts. Saving a prefab requires overwriting as long as it is not directly accessed from the prefab editor interface [10].

The Prefab system in Unity allows for creation, configuring, storing of a game object [2][4] with all its components. If one intends to reuse a game object configured, for example, as a character or a landscape – in different locations within the same scene or in several distinct scenes of the project, it must be converted to a prefab. This is a recommended course of action, and not just the copy/paste procedure for a game object, because the Prefab system allow for automatic keeping of all copies synchronized. In order to create complex hierarchies of objects it is useful to imbricate prefabs into other “container” prefabs, because there is the possibility to overwrite the settings for individual prefab instances. The use of Prefab system is also useful when one needs to instantiate game objects. In order to instantiate a prefab during execution time, the code of the script must contain a reference to that respective prefab. One can create this reference by creating a public variable inside the code for storing the Prefab reference. The public variable from within the code is shown as an attributed field inside the Inspector. Later, we can assign the real prefab that we want to use, inside the Inspector.

Lights and lightning inside Unity game engine

The Unity game developers can create a very realistic lightning that matches very well to a vast array of art styles. In Unity lightning is implemented by approximating the way real light behaves in the real world, because Unity uses very details models about lightning processes for an as realistic result as possible, or simplified models for a more stylized result. The lightning created inside Unity can be direct or indirect and for more realistic lightning results, one must simulate both direct and indirect light sources. The direct light is the light that is emitted, hits a surface and it then reflected directly into a sensor (e.g., a camera). Indirect lightning represents the light that hits different surfaces more times or the sky lightning [2]. Unity can calculate direct illumination, indirect illumination or both direct/indirect illumination and the techniques used by Unity depend on the mode the game creator configures the Unity project settings. Instruments for illuminating the scene use quite easy to configure parameters, as observable from Figure 6.



Figure 6. Panel for setting lightning parameters

The main properties used for defining lightning in Unity are also shown in Figure 7.

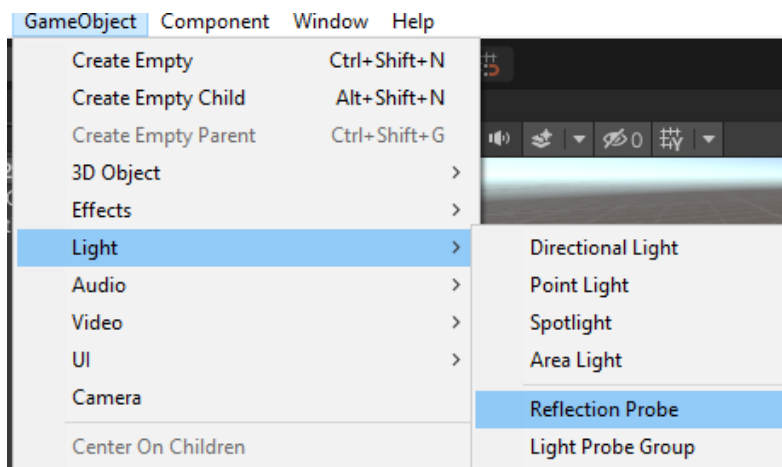


Figure 7. Light types in Unity

Identifying main characteristics for different types of lighting

- Point lights

A point light is localized in space in a certain, individual/single point. This type of light sends the light towards all directions (in a spherical shape) in equal parts, as shown in Figure 8. This correlation is known as the “law of the inverse of the square” and it is similar to the way real light behaves in real world. The point lights are useful for simulating lamps and other local light sources within a scene and can be used to create sparks or explosions for convincingly illuminate the surroundings inside a game environment [2].

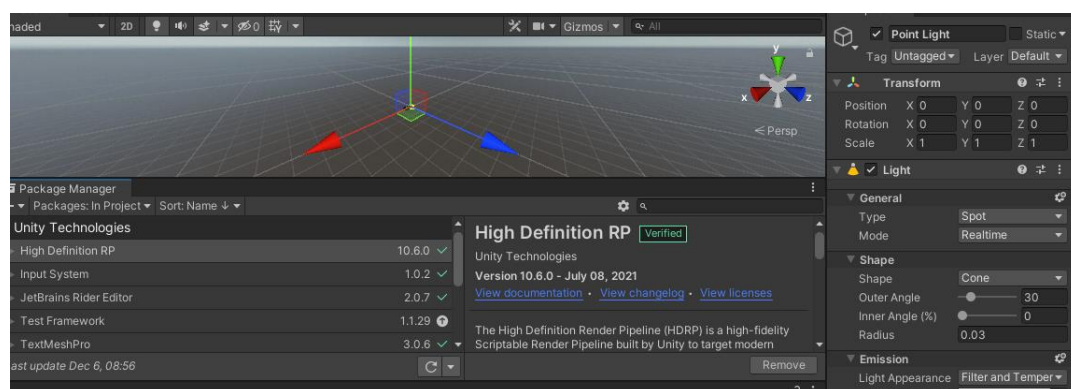


Figure 8. Point Light in Unity

- Spotlight

A spotlight has a specified location and a set interval after which the light goes off. Nevertheless, the spotlight is limited to an angle, which creates a cone-shaped illumination region. The center of the cone indicates the forward direction (Z) of the lighting object. The light diminishes, also, towards the margins of the light cone. The spotlights are used for artificial light sources, such as lanterns, headlights etc [2]. The settings for the spotlight illumination are shown in Figure 9.

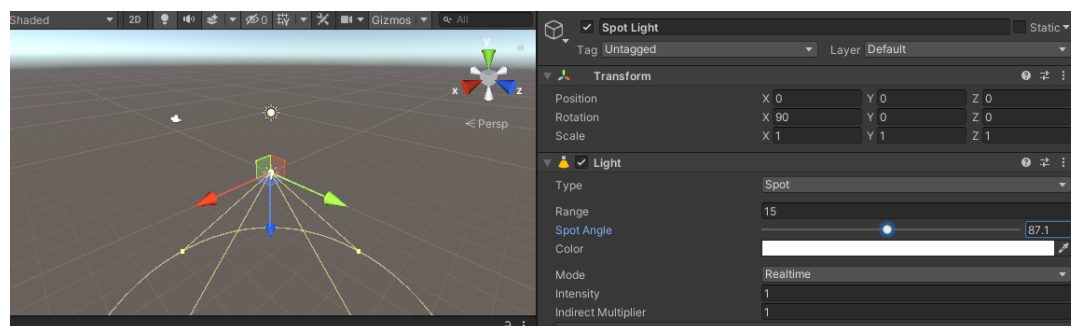


Figure 9. Spotlight in Unity

- **Directional lights**

Directional lights are used to create effects, like the sun light, and can be deployed as far-away light sources, which exist somewhere at an infinite distance from the objects. All objects inside the scene are illuminated like in a scenario where the light always comes from the same and one direction. The distance between the light and the object being enlightened is not specifically defined, thus the light intensity remains unchanged all the time. Directional lights represent big, far-away, sources of light that come from somewhere “outside” of the world envisioned inside the game-development project. Inside realistic scenes, this approach would be used in order to simulate the sun’s or the moon’s light, while in abstract games this type of lights would allow shades addition without specifying an exact source of lighting [2].

By default, each new Unity scene contains a directional light. Rotation of the default directional light (the “sun”) triggers the “skybox” update. When the light would be inclined laterally, but still parallel to the ground-line, the sunset-like effects can be obtained. Moreover, directing the light towards up makes the sky dark, as if the scene time would be night. With the light inclined down, the sky inside the game-development project will resemble normal/real daylight conditions. In case the skybox element is selected by the developer as Ambient source, the Ambient Light will be changed in correlation with its colors (see Figure 10).

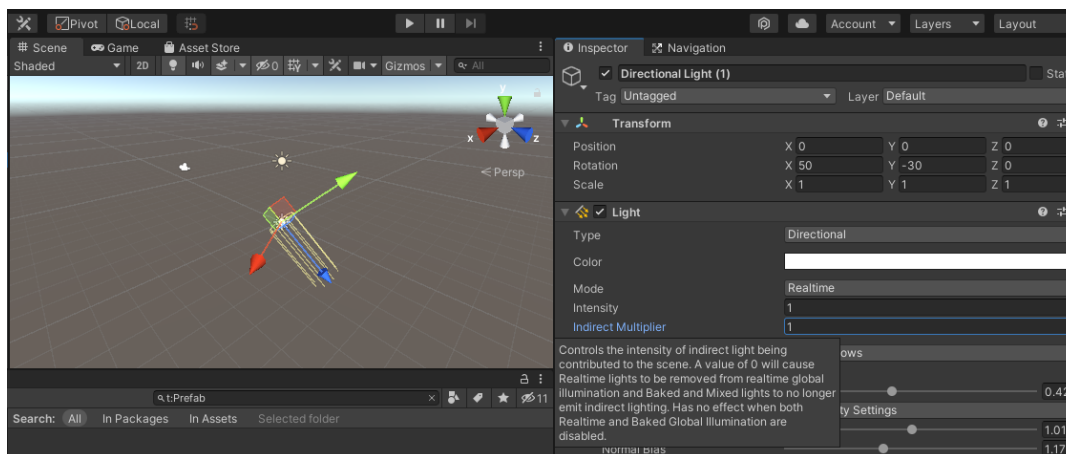


Figure 10. Directional lights in Unity

- **Zonal lightning**

An area-light is in fact defined by a rectangle-shape in space. In this case, the light which would be emitted is spread towards all directions, in a uniform manner on the surfaces, but only from one side of the rectangle. There is no manual control for the range of a lighting area, even if the light intensity will diminish according to the inverse square rule, as it

moves away from the source. The setting of an area light parameters can be seen in Figure 11.

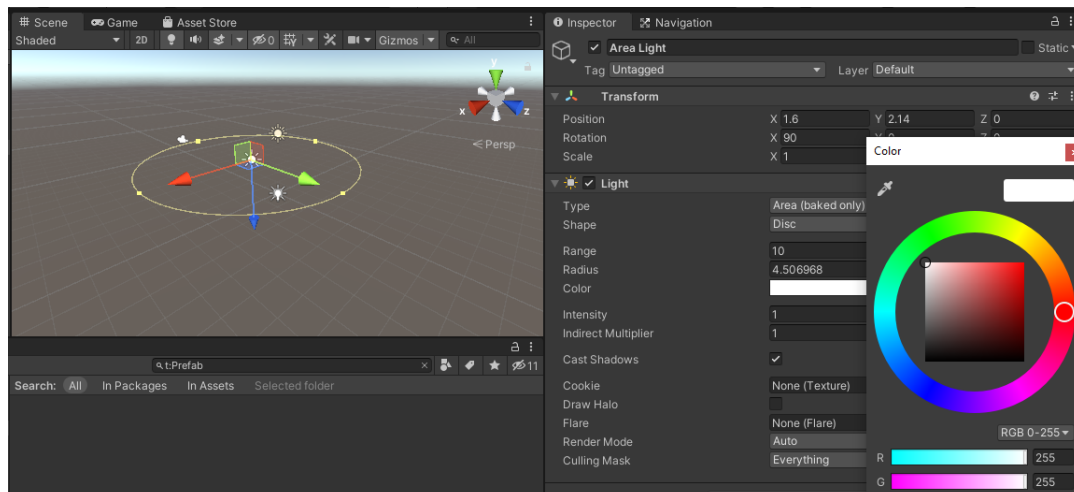
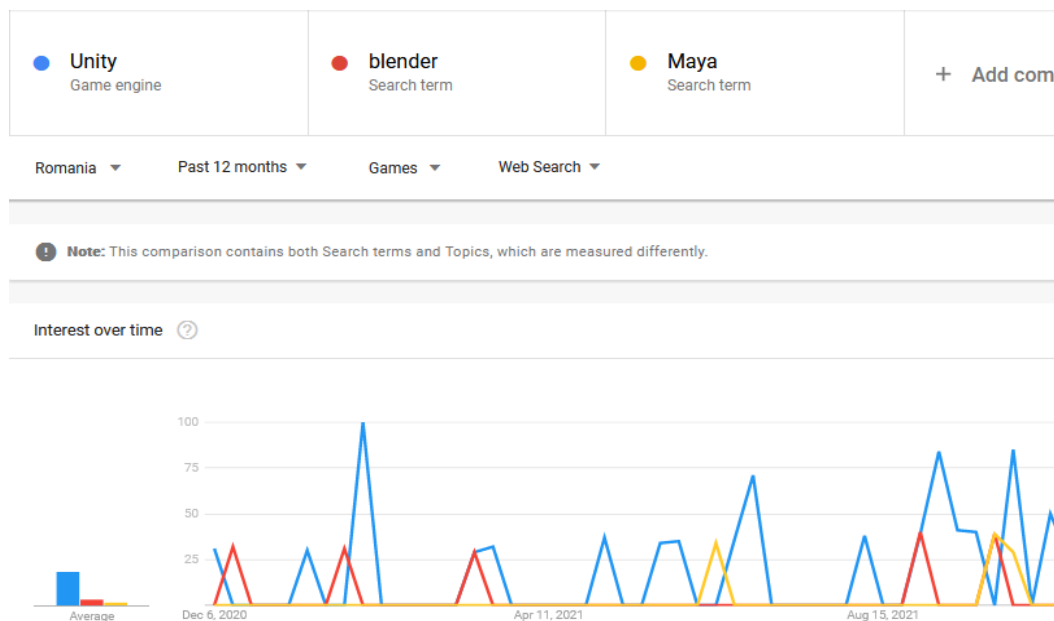


Figure 11. Area Light in Unity

4. Users' attitude to the Unity game engine

Because the evolution of computer/electronic games is a very fast paced one, we were interested in identifying the users' interest, especially within the current pandemic context, towards the game development technologies. By analyzing Figure 12, we can see a comparison of some of the most widely used technologies in this field: Unity, Blender and Maya, from the point of view of the Google engine searches of these elements in Romania. The Unity technology, probably due to its flexibility and intuitive approach, ranks first.



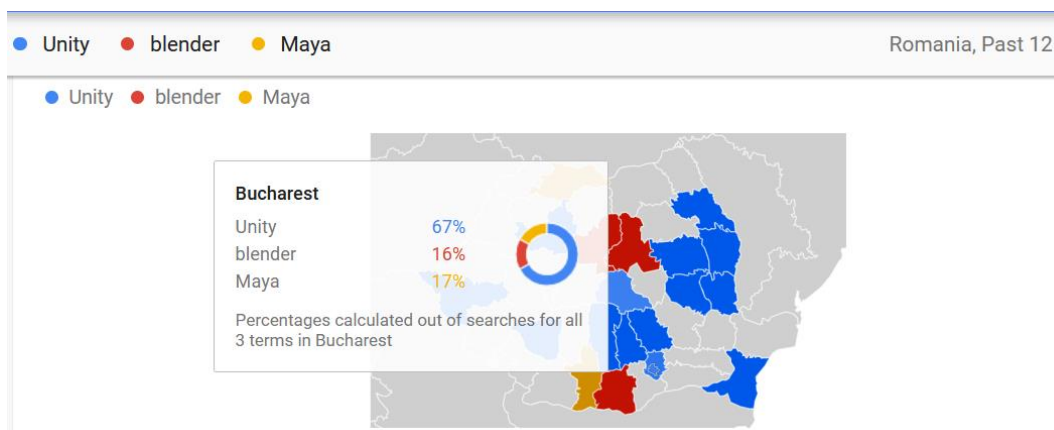
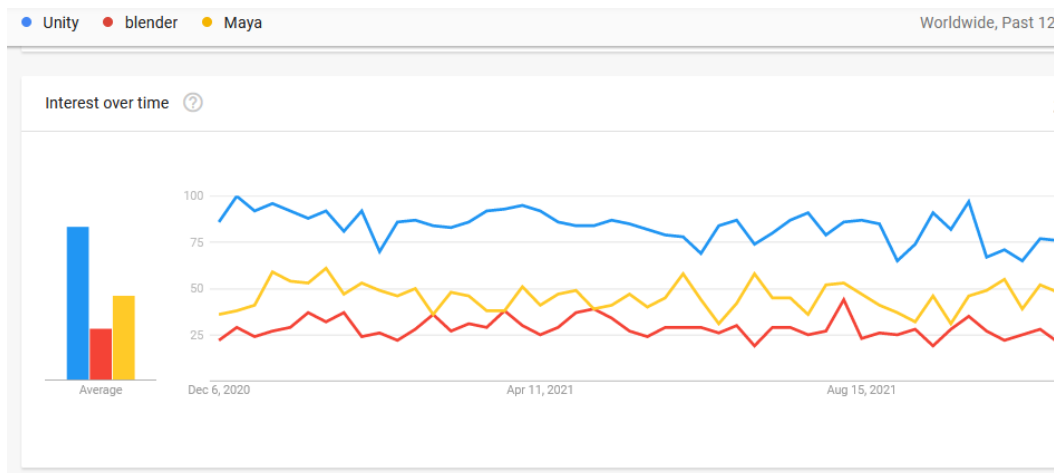


Figure 12. Evolution of Unity, Blender, Maya game development technologies searches according to Google Trends (for Romania)

Inside Figure 13 we can observe the worldwide users' interest towards Unity technology, in comparison with the other two technologies chosen, and the place of Romanian users on this search statistics. Again, Unity is way over the other two technologies, in several points even overpassing the sum of both other variants put together.



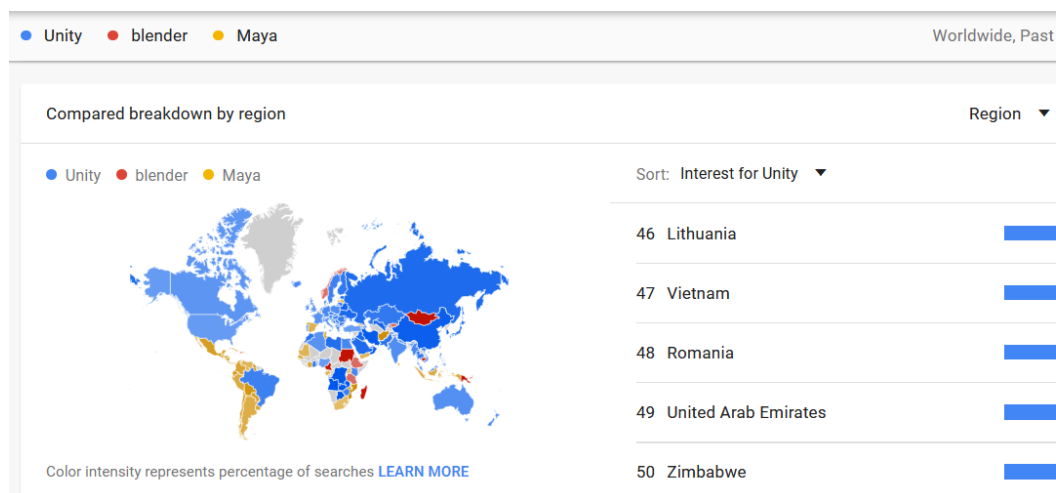


Figure 13. Last 12 months comparison for the Google searches on the Unity, Blender and Maya game development technologies

Considering the Unity Technologies Report of 2021¹²⁴, if we consider the entire niche of applications developed for mobile devices, the Unity engine usage amounts to a staggering 61% of the market. The second most widely used solution, based on native/custom engines is only rated at 15%, and the third option is represented by GameMaker Studio / Unreal / AppGameKit all with a share of 5% (see Figure 14).

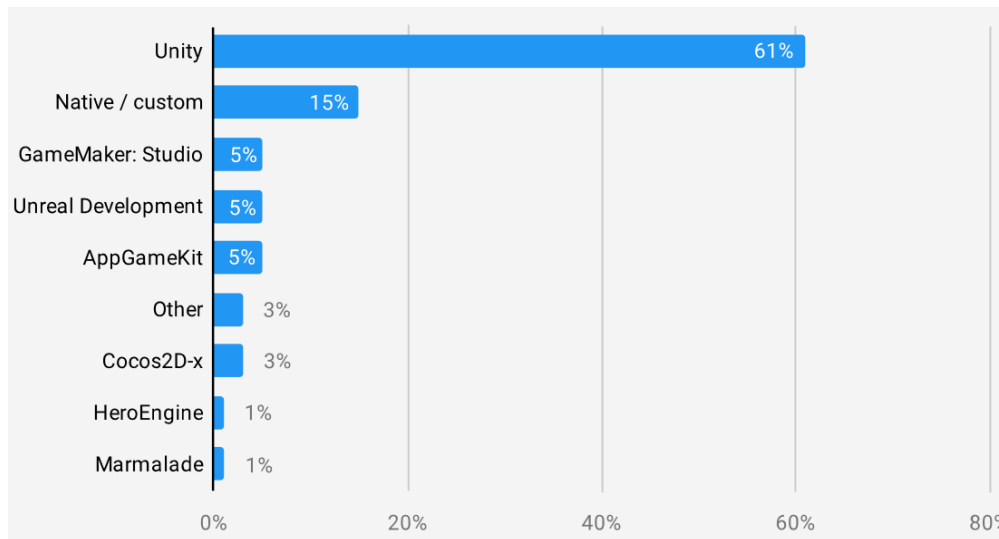


Figure 14. Unity engine market share for mobile devices applications, as of 2021 Gaming Report from Unity Technologies

Going a little bit more in depth with the analysis of the Unity game engine technology usage today, based on the AppBrain¹²⁵ public statistics, we can certainly argue that Unity

¹²⁴ <https://create.unity3d.com/2021-game-report>

¹²⁵ <https://www.appbrain.com/>

is by far the most widely used solution for game development also for the Android apps niche, available through the Google Play store. As we can see in Figure 15, the Unity-based games from the Google Store cover almost 12% of the total number of listed applications in this category. Moreover, the total number of installs is even bigger, amounting to 16.16%.

As a further argument for the prominence of this game development technology, Unity has a market share of over one third of the most important apps in Google Store (top 500, as listed on December 9, 2021).

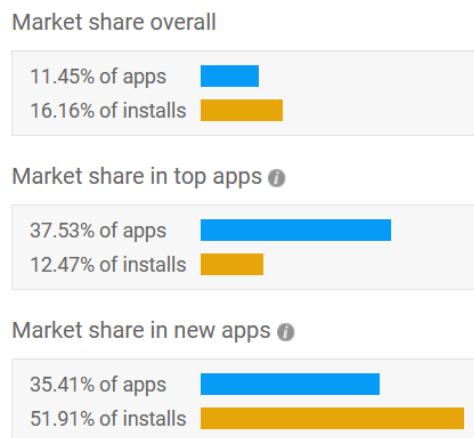


Figure 15. Market share statistics for Unity games in Google Play store¹²⁶

Supporting the previous statements regarding the increased interest in Unity during the current pandemic context, the statistics show a strong Unity-based batch of new applications coming to the Google Play store constantly. No less than 35.41% of the new applications published during the last 30 days¹²⁷ inside the Google Play store were built upon the Unity game engine.

Next, seeing the game platform development from the business point of view – for its users – we have researched the different monetization solutions available to the developers. Within this category of applications there are two main concepts used for monetization purposes: ads and in-app purchases. In Figure 16 we have the picture of the current advertising providers for the mobile applications. Even though the first place is unsurprisingly held by Google and the third place in the medalists' top is also not a surprise with Facebook, the second-place contender is again Unity. The ease of use and seamless integration into the Unity engine, together with the ever-increasing diversity of available advertisements and the new tools and features of the latest Unity engine iterations made it possible for Unity to compete on par with the two greatest companies in the field at this moment.

¹²⁶ <https://www.appbrain.com/stats/libraries/details/unity/unity-3d>

¹²⁷ as of December 9, 2021

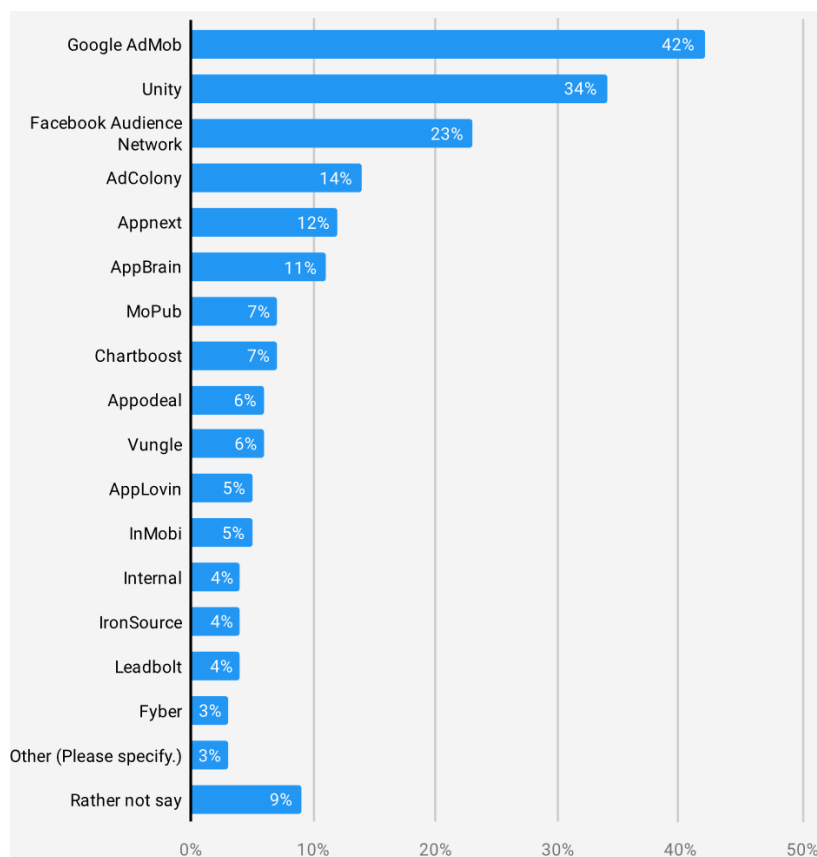


Figure 16. Advertisements providers for mobile applications in 2020, as of 2021 Gaming Report from Unity Technologies

The second solution for monetizing one Unity developed application is through the use on in-app purchases. For this path, Unity offers the possibility of the developers implementing in-app purchases on their own. Nevertheless, all professionals are almost unanimously in accord that this solution would be the best way. Inside the Unity engine environment there are several established plugins, widely used and trusted, that can take care of most technical aspects of the implementation of such a system. These plugins offer all the same basic features, the difference being usually made on five key aspects:

- only including an in-app purchases library or including an entire framework that supports creation of an fully fledged in-game economic system
- platform availability, some plugins being cross-platform while others are targeting only a certain platform
- support, as some plugins have only limited support (at least for the free/entry level versions) while others offer producer's support and/or community support
- market-share, as some plugins are more popular than others
- pricing, as everybody is in the end interested in the best ROI possible for their acquisition of a certain software element to increase future income; some plugins are free, some have free and paid versions, and some are only available as commercial versions.

Among the best plugins available for in-app purchases monetization within Unity we can mention:

- Prime31¹²⁸ – a long-time established plugin, very popular, with different versions for Android and MacOS, offering an in-app purchases library
- Unibill¹²⁹ – a popular in-app purchases library with cross-platform capability, available to handle simultaneously Apple's App Store, Google Play, Amazon, Samsung, and Windows Store
- IronSource¹³⁰ – a cross-platform plugin with extended capabilities as both in-app purchases library and fully fledged SDK for building an entire in-game economy system.

5. Conclusions

Unity offers its users the possibility to create 2D and 3D games, based on a game engine that contains a basic API for scripts written in C#. Unity provides support for bump mapping, reflection mapping, parallax mapping, spatial occlusion on the screen, dynamic shadows using shadow maps but also full-screen post-processing effects. The game developers can become editors within the assets shop and can even sell their creation thorough it. Unity Assets Store can be visited from the Unity website as well as directly from the Unity Game Engine. Today, especially within the pandemic context which overwhelmed the entire world with online/electronic content, there is a clear trend towards using and developing game engine platforms. This fact is supported also by the search terms analysis from the Google search engine for Romania. By comparing the evolution of searches regarding Unity, Blender and Maya game development technologies, as reported by the Google Trends tool, and presented in Figures 12 and 13, we can discern a seasonal trend for Romanian users. This fact is due to the influence of the periods of time during which educational activities are more active.

Bibliography

[1] Tsai, Y.T., Jhu, W.Y., (...), Chen, C.Y., Unity game engine: interactive software design using digital glove for virtual reality baseball pitch training, *Microsystem Technologies-Micro-And Nanosystems-Information Storage And Processing Systems*, 27 (4), pp.1401-1417, Apr 2021

[2] <https://unity.com/learn>

[3] Goldstone, W., *Unity Game Development Essentials*, Packt Publishing Ltd., ISBN 978-1-847198-18-1, UK, 2009

¹²⁸ <https://prime31.com/>

¹²⁹ <http://outlinegames.com/>

¹³⁰ <https://www.is.com/>

- [4] Menard, M. and Wagstaff, B., Game development with Unity. Nelson Education, 2015
- [5] Valcasara, N., Unreal Engine Game Development Blueprints. Packt Publishing Ltd., 2015
- [6] Hocking, J., Unity in Action: Multiplatform Game Development in C# with Unity 5, 1st Edition, Manning Publications, 2015
- [7] Murray, J., C# Game Programming Cookbook for Unity 3D, CRC Press, 2014
- [8] Thorn, A., Pro Unity Game Development with C#. Berkeley, CA: Apress, ISBN: 978-1-4302-6745-4, New York, 2014
- [9] Dickson, P.E. et al., An experience-based comparison of unity and unreal for a stand-alone 3D game development course. In: Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education, pp. 70–75, 2017
- [10] Blackman, S., Beginning 3D game development with Unity: World's most widely used multi-platform game engine, Apress, 1st Edition, ISBN: 978-1-4302-3423-4, 2011.