We are IntechOpen, the world’s leading publisher of Open Access books
Built by scientists, for scientists

6,600
Open access books available

178,000
International authors and editors

195M
Downloads

154
Countries delivered to

Our authors are among the

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities

WEB OF SCIENCE™
Selection of our books indexed in the Book Citation Index
in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com
Chapter

The Role of Game Engines in Game Development and Teaching

Branislav Sobota and Emília Pietriková

Abstract

This chapter explores the essential role of game and graphics rendering engines in creating computer games and their applications in education. Starting with a brief discussion on the importance of visualization in game development, we analyze popular game engines, including CryEngine, Irrlicht, Unreal Engine, Unity, and Godot, and graphics rendering engines (cores), including OpenSceneGraph (OSG) and Object-Oriented Graphics Rendering Engine (OGRE). Each engine’s unique features and pedagogical potentials are discussed, with emphasis on Unity, Godot, and Unreal Engine’s interactive and step-by-step tutorials. This analysis reflects the adaptability of game engines to different teaching styles and provides insights into their effectiveness as tools for teaching game development and computer science. The chapter aims to highlight game engines’ role in game creation and education by exploring these topics.

Keywords: game engines, teaching game development, computer science learning, visualization, game engines tutorial

1. Introduction

Computer games and, of course, especially their development, are a significant part of computer science. Modification or change of some procedures in game development is necessary due to modern technologies. In this sense, the development of technical means, especially graphics adapters, is advancing, for example, the possibilities of computer graphics as an essential component of computer games. On the other hand, the opportunities and problems of graphics libraries and engines define the need and maximization of graphics adapters’ use. This aspect explains a significant synergistic connection between computer science and computer games. This connection is also reflected in the basic architecture of the currently standard gaming computer system (Figure 1).

As a person perceives up to 80% of information by sight, it is understandable that the visualization subsystem is dominant in this case. And the central phenomenon of the visualization subsystem is the use of computer graphics. Its usage can be seen at distinct levels of the hierarchy (Figure 2): starting with graphics software at the machine code level and ending with complex graphics engines. To this, it is necessary to add the graphic options of higher programming languages/browsers use,
graphic libraries (e.g., OpenGL\(^1\), (Vulkan\(^2\)), (webGL\(^3\)), graphic frames (e.g., ThreeJS\(^4\), BabylonJS\(^5\)), or medium-level graphic cores (e.g., OpenSceneGraph\(^6\), A-frame\(^7\)).

It is crucial to present this relationship in this way, also when teaching IT (Information Technologies) students. Including all the mentioned levels in the teaching introduces individual levels to students and shows the increase in the possibility of using computing technology in this area. In addition, increasingly higher levels of hierarchy expand the circle of users, especially game developers, beyond the exact group of IT specialists. In this context, virtual reality (VR) and its advanced technologies also play an essential role.

\(^1\) OpenGL, home page: [cit. 2023-03], URL: http://www.opengl.org/
\(^2\) Vulkan, home page: [cit. 2023-03], URL: https://www.vulkan.org/
\(^3\) webGL, home page: [cit. 2023-03], URL: https://get.webgl.org/
\(^4\) ThreeJS, home page: [cit. 2023-04], URL: https://threejs.org/
\(^5\) BabylonJS, home page: [cit. 2023-04], URL: https://www.babylonjs.com/
\(^6\) OpenSceneGraph, home page: [cit. 2023-03], URL: http://www.openscenegraph.org/
\(^7\) A-Frame, home page: [cit. 2023-04] URL: https://aframe.io/
The Role of Game Engines in Game Development and Teaching
DOI: http://dx.doi.org/10.5772/intechopen.1002257

When teaching game design and development, introducing game theory can enhance students’ understanding of game mechanics, dynamics, esthetics, balance, etc. Utilizing game engines in the classroom provides an excellent platform for practically applying game theory concepts. For instance, students can use game engines for prototyping and testing different game mechanics while observing the result of various strategies and interactions and effectively experiencing game theory in action. While this chapter focuses on the role of game engines in teaching game design and development, the interplay between these engines and game theory underpins the educational potential of these tools. As students learn to use game engines, they inherently deal with game theory concepts, bringing abstract mathematical models to life.

That is why the Department of Computer and Informatics of the Faculty of Electrical Engineering and Informatics (FEEI) at the Technical University of Košice has been running the Design and Development of Computer Games course and related subjects (e.g., Computer Graphics or Virtual Reality Systems) for several years. As we assume that IT (Information Technologies) students are potentially future creators/programmers of computer games, they must have at least a basic understanding of game theory to be successful in the implementation. Therefore, as a part of the educational process, students become familiar with various methodologies and technologies related to the design and development of computer games, such as high concept, market research, storytelling, prototyping, balancing rules, Mechanics-Dynamics-Esthetics (MDA), or iterative game development.

Game engines are integral to various domains, including game development, VR, and computer science. The creation of computer games is a sophisticated process that requires artistic vision and robust technology to bring that vision to life. The essential tools used in this process are game engines, handling many necessary elements, such as rendering graphics, processing physics, and managing inputs. In this chapter, we explore the crucial role of game engines in developing computer games and learning.

Following this, we analyze several popular visualization cores and game engines, presenting their unique features, strengths, and weaknesses. In the latter part of the chapter, we shift our focus to the educational implications. We illustrate how different approaches can influence learners through practical tutorials in Unity\(^{26}\), Godot\(^{21}\), and Unreal Engine\(^{20}\). Our exploration underscores the value of practical, applied learning in equipping students with industry-relevant skills. We aim to highlight the role of game engines in game creation and education.

We will also present the outputs of some activities recorded during the educational process with students in the third year of the Informatics bachelor’s degree in a full-time study form. We attach significant importance to the fact that these are real procedures and activities processed over several years and can thus provide a basis for further studies. Therefore, we believe the experiences presented in this chapter will find their readers.

2. Visualization subsystems in games

In general, the purpose of the game’s graphics or visualization subsystem, in particular, is to support the loading and rendering of the scene for visualization. It defines the data format for the scene and handles user input. There are additional features in systems such as special effects and physics support, and many computer features include an integrated environment for scene development and scripting. The
systems with extended functionality are also called 3D (three-dimensional) game engines if their primary purpose is to support game development.

Among the functions that can be associated with the visualization subsystem of the game concerning its implementation are:

- **License.** The license must allow the software to be used for commercial purposes.

- **Software extensibility and source code availability.** There should be an option to modify the selected candidate using its application programming interface (API), but the best option would be to have its source code available for customization.

- **Quality of documentation.** Documentation should be complete, understandable, and up-to-date.

- **Support for standard 3D modeling formats.** They determine the possibilities of importing 3D models created using other software or scanned using 3D scanners.

- **Support for the latest 3D graphics features.** It is vital for creating the most realistic video output possible.

- **Programming languages available for scripting.** Scripting, or so-called soft coding, is a way to program visualization controls without compiling. In the case of commercial products, this is often the only way to communicate with the core at the programmatic level.

- **Support for advanced virtual reality hardware solutions.** In this sense, focusing more on supporting stereoscopic video output and sensing user movements is necessary.

- **Availability of development tools for scene creation and scripting.** The availability of such a development tool (editor or integrated development environment (IDE)) can significantly reduce the time required to prepare a visualization scene.

- **Parallelization options.** Rendering high-quality video output requires a powerful computer system. Within game systems, we often need to display more than one output; for example, two images are necessary for stereoscopic display. In addition, immersive solutions often contain multiple displays (for example, front, back, left, right, top, and bottom in the case of a virtual cave). To maintain high quality, the images must be rendered in parallel.

In addition to these parameters, prototyping and game testing are also important. If we talk about prototyping, we recognize three types of game prototypes [1]:

- **Software.** Prepared in a game engine like Unity.

- **Physical.** Physical interaction and activity with the playtester.

- **Paper.** Board game, the most common type.
In the case of testing, the literature states that four types of testing are crucial to the design and development of computer games [2, 3]: Focus Groups, Quality Assurance (QA) Testing, Usability Testing, and Playtesting.

Next, we will provide an overview of some available visualization cores or complex game engines, both free solutions and commercial ones.

2.1 Visualizing and manipulating the game world

Something has already been hinted at visualization earlier, and of course, much more will be written about it, as it is currently a key part of a computer game. Therefore, let us mention here that, from a technical point of view, visualization is possible using various technologies. Manipulation of objects is also possible here. For interactive manipulation of objects in the game, it is possible to use classic input devices like a keyboard and mouse or various position sensors. The overall flow of data between individual structural elements from the point of view of working with the game world is depicted in Figure 3.

The visualization subsystem of the game provides the central part of the information for the player in the game environment. This subsystem calculates the image for the player in the game world. For visualization, the visualization core mentioned above is often used in games. The visualization core is usually based on the serial use of information, and currently, the graphics processing unit (GPU) (graphics processor at the hardware level) services are used. On the input side, there is a model of the game world, and on the output side, there are correctly colored pixels on the display. Between these two stages, other stages partially change the input information. The main idea for implementing the game visualization subsystem is to minimize the time used in each stage for the real-time system, which is hugely often needed in games. In multiprocessor systems, including General-Purpose Graphics Processing Unit (GPGPU) technology, it is possible to apply some parallel stages, representing a significant speedup.

Figure 3.
Basic data flow in a computer game.
Complex game engines, at the highest level of the hierarchy (Figure 2), often support visual programming in the form of WYSIWYG (What you see is what you get) technologies (e.g., Blueprints for Unreal Engine). WYSIWYG (What you see is what you get) and other interactive tools thus simplify the entire development of the game and, in principle, bring this process closer even to people without an exact IT education. For example, the following features are available:

- Integrated graphic editors of the scene and dynamics.
- Integration of programming and scripting languages.
- Integration of support for different platforms and operating systems.
- Integration of support for various technical means, including network communication or parallelization of calculation.

3. Visualization cores and game engines

In this section, we will provide an overview of some available visualization cores and complex game engines, both free solutions and commercial ones, which are used or have been used as part of the courses at the authors’ affiliation.

Of the freely available solutions, OpenSceneGraph\(^6\) is interesting, which is not the easiest to understand and integrate but supports parallel calculations and modern VR technologies. Many implementations in the world form an excellent knowledge base. These factors currently make it the favorite in this category for the need to integrate graphics systems into game solutions unrestrictedly.

The commercial solutions mentioned next support the most modern imaging techniques and provide limited options for integrating modern input 3D interfaces (for example, based on virtual reality). However, compared to freely available solutions, they enable the quick creation of scenes thanks to integrated editors, which are very well documented. On the other hand, the limited possibility of modifications, only sometimes available source code (or at least not at an affordable price), and no or limited support for parallel computing, could be a limitation in distributed gaming environments.

3.1 OpenSceneGraph (VulkanSceneGraph project)

OpenSceneGraph\(^6\) (OSG, Figure 4) is a freely available high-performance 3D visualization core written in the C++ programming language (not to be confused with OpenSG\(^8\), which is no longer in development and represents an entirely different scene graph API, somewhat similar to OpenGL Performer). It uses the OpenGL\(^1\) library (the VulkanSceneGraph Project\(^9\) is available for the Vulkan library\(^2\)). It is multiplatform and can be used on MS Windows, OS X, and GNU/Linux platforms. Since version 3.0.0 (3.6.5 currently), it is available also for mobile platforms, Android and iPhone Operating System (iOS). It is also used as the basis for several virtual reality solutions, including JuggLua VR, CalVR, and Vrecko.

---

\(^6\) OpenSG, home page: [cit. 2022-06], URL: https://sourceforge.net/projects/opensg/
\(^8\) OpenSG, home page: [cit. 2023-03], URL: https://vsg-dev.github.io/vsg-dev.io/
\(^9\) VulkanSceneGraph, home page: [cit. 2023-03], URL: https://vsg-dev.github.io/vsg-dev.io/
Scenes are represented by a data structure called a scene graph, which organizes a logical and often spatial representation of the scene. The documentation is not OSG’s strong point. It contains several user and programmer guides. Most used 3D formats are supported through plug-ins that are part of the OSG core distribution. OSG effectively implements the best graphic techniques, such as early elimination of the primitives based on overlap and visibility, progressive simplification of geometric detail of models, sorting of display states, and so-called particle systems. Support for graphics chain programming (shader) is implemented based on OpenGL. Of the analyzed solutions, OSG has the longest history of use in games or visualization systems. Support for stereoscopic visualizations and peripheral input devices has been implemented. There are several extensions, for example, osgTerrain (terrain rendering), osgAnimation (character and rigid body animation), osgManipulator (3D interactive controls), osgShadow (shadow framework), osgParticle (for particle systems), osgFX (special effects framework), or osgVolume (high-quality volume rendering (with Dicom plug-in for support of medical datasets)). OSG directly supports parallel computing because the scene graph supports multiple graphics contexts and stores graph states in fast local memory. Another option for parallelization is using the program Equalizer tool\(^\text{10}\).

3.2 Ogre

OGRE\(^\text{11}\) (Figure 5) is a scene-oriented 3D core. It is created in C++ object programming, and the abbreviation is made from the Object-Oriented Graphics

\(^{10}\) Equalizer, home page: [cit. 2023-01], URL: http://www.equalizergraphics.com/

\(^{11}\) Ogre3D, home page: [cit. 2023-01], URL: http://www.ogre3d.org/
Rendering Engine. It is a pure displaying core, so support for playing sounds, physics, and other features is at the plug-in level. Its expandability is guaranteed by a highly modular architecture based on plug-in modules. It supports Direct X\textsuperscript{12}, OpenGL\textsuperscript{13}, and Vulkan\textsuperscript{14}, and it is distributable not only to MS Windows, Mac OS X, and Linux but also to Android and iOS platforms. Its highly rated feature is excellent support for animations based on skeleton models.

Object-Oriented Graphics Rendering Engine (OGRE) is freely redistributable with open-source code, so price and licensing are no issues. It is very well documented, and in addition to a rich set of freely available sources, there are two books [4, 5]. OGRE directly supports only its model format, but fortunately, there are export libraries in plug-in models to several 3D editors. The support of advanced graphic standards is at a reasonable level. The high-level shader language (HLSL) supports graphics string programming. The scripting is not directly available; however, some extensions add this capability. The support for stereoscopy is again provided by a unique project called Stereo Vision Manager. The VRPN\textsuperscript{13} (Virtual Reality Peripheral Network) library can support advanced 3D interfaces. Furthermore, some extensions make it easier to integrate this solution. OGRE does not include direct support for parallelization, but there are several attempts for this extension, again as in the case of OpenSceneGraph\textsuperscript{6} using the Equalizer tool\textsuperscript{10}.

3.3 CryENGINE

CryENGINE\textsuperscript{14} (Figure 6) is a professional 3D game engine. It has been used in the creation of several highly rated game titles. In addition to the game world, virtual training for American army soldiers is also based on it. It is available for MS Windows and Xbox, Playstation, and Wii game platforms. It also has direct support for playing sound effects and physics.

The users can use this 3D engine within the academic sector without fees. For commercial use, it is necessary to pay a fee, which depends on the purpose of the developed title, and it is determined based on communication with the manufacturer. The core is written in C++ and can be extended via API or scripting. The documentation is of a high standard, as is customary with commercial products. It supports only

![Visualization samples based on CryENGINE.](image_url)
The internal format of 3D models but for the most used 3D editors, such as 3D Studio Max or Blender, official exporters, in the form of plug-ins and extensions. This core, built on DirectX\textsuperscript{12} in all its versions, has always supported the current state-of-the-art display techniques. There are two scripting methods: either using the LUA\textsuperscript{15} language or a visual scripting editor called FlowGraph. The stereoscopic display is directly supported. There must be a known implementation of support for modern 3D interfaces yet. A sophisticated editor called Sandbox is used to create the scenes. The parallelization is not directly supported, but the synchronization of multiple computers running this core can, in principle, be implemented (this was done in the project mentioned in [6]).

3.4 Irrlicht

Irrlicht\textsuperscript{16} (Figure 7) is freely distributable with open-source C++ code. It runs on multiple platforms. These are primarily MS Windows, Mac OS X, and Linux, but other distributions exist. It has been a stable engine in the works for 20 years. The engine was created in 2003 by a single developer, Nikolaus Gebhardt. Currently, the development team consists of approximately 10 members. Both DirectX\textsuperscript{12} and OpenGL\textsuperscript{1} can be used for display, as well as internally developed software rendering. One of the main advantages of this engine is no requirement for the installation of additional third-party libraries, making it quite easy to install and run.

Since Irrlicht is freely distributable and can be used commercially, the source files may be modified, extended, and redistributed without a charge. It is well documented, including a description of the programming API, and it includes tutorials with examples for straightforward and advanced use cases. Two books also deal with its description and examples [7, 8]. Irrlicht has the broadest support for input 3D formats compared to other solutions. Importing outputs from the editors, such as 3D Studio Max, Maya, and even forms of different 3D engines, e.g., OGRE, is possible. The scripting has yet to be officially supported, but again there are solutions created by the community that add this functionality for languages such as Ruby\textsuperscript{17}, Python\textsuperscript{18}, Lua\textsuperscript{15}, or Perl\textsuperscript{19}. However, some of them are problematic, and support for their

![Figure 7.](image)

Visualization samples based on IrrLicht.

\textsuperscript{12}LUA, the programming language, home page: [cit. 2023-01], URL: http://www.lua.org/
\textsuperscript{13}Irrlicht, home page: [cit. 2022-09], URL: https://irrlicht.sourceforge.io/
\textsuperscript{14}Ruby, home page: [cit. 2023-01], URL: http://www.ruby-lang.org/en/
\textsuperscript{15}Python programming Language, home page: [cit. 2023-01], URL: http://www.python.org/
\textsuperscript{16}Perl, programming language, home page: [cit. 2023-01], URL: http://www.perl.org/
development has already ended. Irrlicht includes a free, extensible 3D editor called *irrEdit*. The parallelization is not directly supported but can be implemented in principle as the source files are available.

### 3.5 Unreal Engine

Unreal Engine\(^{20}\) (UE, now UE5 *(Figure 8)*) was developed by Epic Games more than 24 years ago as a first-person shooter (FPS) system. Unreal Engine has evolved to support many other genres, including diverse types of games and even linear video production, such as animated stories and 3D VR walkthroughs. It should come as no surprise that, for example, instructional designers are using it to create enterprise virtual reality applications. UE is a commercial solution, the use of which is the same as that of Unity. Unreal Engine supports the deployment of projects on multiple platforms, including MS Windows PC, Mac OS X, iOS, Android, AR, VR, Linux, and HTML5, as well as on head-mounted displays (HMDs) and game consoles. The Unreal Engine editor runs on MS Windows, OS X, and Linux. Due to its more intensive use, we will describe it in more detail.

When programming in UE5, there are two main ways to implement logic into games/experiences: the Blueprints visual scripting language and the more traditional C++ programming language. Unlike Blueprints, C++ can be a bit obscure as it can take some time to learn the required syntax, but C++ provides better access to some of the engine’s hidden features. Blueprints also offer an easier way to migrate work from one project to another. Within UE5, several additional tools and toolkits are available for the tasks such as animation, creating and modifying game environments, and multiuser (collaborative) editing.

![Figure 8](https://www.unrealengine.com/)

*Figure 8.
Material creation in the Unreal Engine.*

---

\(^{20}\) Unreal Engine, home page: [cit. 2023-03], URL: https://www.unrealengine.com/
A UASSET file is an asset file used by the Unreal Editor, a game-level editor, part of the Unreal Engine. It contains assets such as Game Levels, Particle Systems, Blueprint Scripts, Materials, Movie Sequences, Textures, Static Meshes, Skeleton Meshes, Light Profiles, or soundtracks used in the game. All objects/models from the Unreal Engine environment are available as a UASSET file.

Users can create objects/models outside the Unreal editor using tools such as Maya, 3D Studio Max, or Photoshop. They then import these assets into Unreal Editor, which converts the objects/models into UASSET files.

In addition to excellent visualization capabilities, Unreal Engine 5 provides functions to support multiple players/users (client-server model), especially in multiplayer games. Overall, Unreal Engine includes a robust networking framework that is also used by some of the most popular online games in the world to help streamline the process. In multiplayer/user mode, Unreal Engine uses the already-mentioned client-server model. One device on the network acts as a server and hosts a multiplayer game session, while all other players’ devices connect to the server as clients. The server shares game state information with each connected client and provides a means for them to communicate. The server, as the host of the game, has only the actual and authoritative state. In other words, multiplayer games take place on the server. Each client remotely controls their game instance to the server (Pawn) and sends procedure calls to perform game actions on the client. However, the server does not transmit visual information directly to client monitors. Instead, the server replicates game state information to each client, telling it what actors exist, how those actors should behave, and what values various variables should have. Each client then uses this information for the visualization itself.

Unreal Engine is currently free to download and use. One must pay after the game or app is released. The fee is from the profit after earning the first $3000 per product per quarter. It is also possible to download the complete UE source code and modify it according to the needs of the project being created. As mentioned earlier, users can write new classes and parts of the user interface in C++. UE Blueprints are usable for scripting game logic using nodes.

As part of solving various projects at the authors’ home department, UE was used, for example, to create a virtual-reality therapeutic system for paretic patients [9] (Figure 9).

3.6 Godot engine

The Godot engine[21] (Figure 10) is an open-source project under the MIT license, making it available for free download, use for personal and commercial purposes, modification, repair, or extension of its source code. This is also one of the reasons why this engine is a viable choice for students and beginners.

The Godot engine is cross-platform, allowing the game’s creation for Windows, Linux, MacOS, mobile phones, web browsers, and consoles, although the support here is weaker. Both 32- and 64-bit versions of Windows and Linux or MacOS are supported. Godot supports two-dimensional (2D) and 3D game development, but it excels especially in 2D, as mentioned by Dealessandri[22]. A 3D game creation is possible but more complicated.

Multiple programming languages are supported. First, the GDScript scripting language is specially created for the engine’s needs. The syntax reminds us of Python;

---
[21] Godot Engine, home page: [cit. 2023-02], URL: https://godotengine.org/
however, it is closely adapted to the engine, increasing its optimization. Its main advantage is the reduction of development complexity. In addition to GDScript, Godot supports C#, C++, and VisualScript.

Its developers try to keep the core of its functionality small. The effort is to add functionalities as plug-ins. One of the reasons is to have as minimal maintenance as possible for code that constantly needs to be tested. Furthermore, it makes the compilation of games faster. Thanks to its smaller and cleaner code, this makes contributing to the engine project easier. And thus, the size of the binaries of the editor and the exported projects is kept small.

The Garden Path\(^ {23} \) is an exciting game created in the Godot engine. It is a 2D top-down adventure game. Another game is Resolutiion\(^ {24} \). It is an esthetically pleasing 2D top-down action adventure with a pixel art style. The Human Diaspora game\(^ {25} \) is an excellent example of how 3D projects can be created in the engine. It is a 3D First-Person Shooter (FPS), i.e., a shooter with a first-person view. The Dauphin game is also enjoyable. It is a 2D Role-Playing Game (RPG), i.e., a game in which the player

\(^{23}\) URL: https://store.steampowered.com/app/1638500/The_Garden_Path/ [cit. 2023-05].
\(^{24}\) URL: https://store.steampowered.com/app/975150/Resolutiion/ [cit. 2023-05].
\(^{25}\) URL: https://store.steampowered.com/app/1395420/Human_Diaspora/ [cit. 2023-05].
stands in the role of the character he is playing and gradually builds him up and increases his abilities.

One very high-quality and well-crafted instruction source is the GDQuest channel on YouTube. This channel also has a webpage offering paid courses in Godot. We recommend [10] from the books.

3.7 Unity Engine

Unity26 (Figure 11) is a complex game engine. It represents a commercial solution. It is among the most popular game engines due to its ease of use and the wide range of functionalities it offers. Unity is a cross-platform game engine developed by Unity Technologies. As mentioned, it ranks among the most popular game engines, as it is suitable for beginners and offers a wide range of tools for 2D and 3D projects. Among these tools, we can include collision detection, 3D rendering, effects, physics simulation, or a simple editor. The main advantages of Unity include a large community of game developers, a lot of tutorials, and easy export of the game to multiple platforms. Unity is especially popular with smaller and medium-sized projects, as it allows for rapid prototyping and works on a system of adding components to game objects, thanks to which it is possible to lay out and change the scene quickly. Supported platforms include, for example, MS Windows, Mac OS, Linux, Xbox, Wii, and Playstation, as well as Android and iOS mobile solutions. Depending on the target platform, Unity supports both DirectX12 and OpenGL1.

Unity has a graphical environment adapted for easy game creation like other game engines. It consists of the Editor (Figure 11 right), consisting of several windows and panels, each with unique functionality. In the image below, we can see the default layout of these windows. Basic windows include:

- **Hierarchy**—It serves to display all objects on the scene.
- **Inspector**—We can edit and add properties of the selected object.
- **Game view and scene**—It serves to simulate the game and edit the scene.
- **Project**—It contains all the necessary files that we use in the project.

![Figure 11. Example of the simple game in UNITY and visualization in the integrated Unity editor.](image-url)
Essential elements of the engine include prefabs, colliders, physics, an audio system, and animations. Prefabs are used as game objects with preset properties and components, thanks to which we can use them in several scenes and easily duplicate or instantiate them. Colliders are used to detect collisions between objects. Unity also includes a physics engine that simulates gravity, acceleration, and other physical phenomena. These tools make it easy for game developers to create prototypes and focus on developing specific parts of the game.

From an educational perspective, several tutorials explain how to work with Unity. Some points are directly taken from this engine's developers or professionals who create various video tutorials.

### 3.8 Teaching and learning with game engines

Game engines not only offer students different pathways for learning but also provide educators with various teaching methods. Each game engine's unique functionality and design can support different teaching styles [11, 12].

CryEngine's flowgraph system offers a graphical scripting interface, which could support a visual or demonstrative teaching style. Educators can visually illustrate game mechanics [13], providing an intuitive way for students to understand game logic without the necessity of text-based coding. On the other hand, Irrlicht, as a lightweight and easy-to-use graphics engine, could support a direct teaching style where educators focus on teaching the underlying principles of 3D graphics without the complexity or distraction of more extensive features [14]. Its simplicity could be especially beneficial when teaching beginners or focusing on foundational concepts.

With its advanced features, Unreal Engine is well suited to a teaching style where the educator imparts detailed knowledge on complex topics like advanced graphics, physics simulation, or artificial intelligence (AI) programming. Its powerful visual scripting system, Blueprint, also provides an avenue for teaching programming concepts more visually and intuitively, supporting diverse learners [15]. For learners aiming to grasp the technicalities of game physics and lighting, Unreal provides a highly illustrative platform.

Godot's unique scene system lends itself to a more systematic teaching approach. Educators can build lessons around the node and scene structure, teaching individual concepts and showing how they fit into a larger whole. This modular approach could support a scaffolded teaching style, where each new concept builds on the previous one [16]. The engine's node and scene structure encourage a modular approach to game design, which can facilitate a deep understanding of object-oriented principles, design patterns, and aspects of game development like the parallax effect.

Unity's user-friendly interface and real-time feedback can support a more facilitative teaching style, allowing educators to guide students through hands-on experimentation. The immediacy of seeing the effects of code changes in the game world provides tangible feedback that reinforces learning. Unity's user-friendly interface and robust community support make it accessible to beginners [17]. Unity's strong community and extensive online resources also provide ample material for independent study, supporting a flipped classroom approach where students explore content outside of class and use classroom time for discussion and application.

Game engines offer educators a range of possibilities to align with their preferred teaching styles. Educators can create engaging, effective learning experiences that
The Role of Game Engines in Game Development and Teaching
DOI: http://dx.doi.org/10.5772/intechopen.1002257

resonate with their students by choosing an engine that complements their approach.

Table 1 outlines each engine’s general teaching approaches and learning styles.

Table 1 illustrates the diverse teaching opportunities presented by different game engines, making them adaptable to various teaching styles and educational contexts. In the following section, we will delve deeper into the educational application of game engines. We will present different tutorials, each adopting various teaching approaches using Unity, Godot, and Unreal Engine. These examples will demonstrate how these engines can be tailored to facilitate different teaching styles, cater to varied learner preferences, and ultimately foster effective learning experiences in game design and development.

### 4. Educational experience with game engines

Our institution focuses on teaching game creation within one academic course and partially in the next four courses. The lectures are devoted to the most critical aspects related to the design and development of games. The laboratories focus on technology and teamwork. As a part of these laboratories, we decided to show students work in three popular engines: Unity, Unreal, and Godot, working with each differently.

For each tutorial, we dedicated two laboratories. During the first laboratory, we introduced the game engine and started with the tutorial, helping students with troubleshooting. This initial introduction was crucial in assisting students to navigate the engine environment and understand the basics. During the second laboratory, we let students work on the tutorials independently. Then, the students finished their work at home.

---

Please note that this table serves as a guideline rather than a strict rule, as these engines’ versatility allows various teaching methodologies to be employed based on the specific course objectives, student backgrounds, and teacher preferences.

---

### Table 1

<table>
<thead>
<tr>
<th>Engine</th>
<th>Teaching approach</th>
<th>Learning style</th>
<th>Suitable for</th>
</tr>
</thead>
<tbody>
<tr>
<td>OSG</td>
<td>Detailed, methodological</td>
<td>Analytical learners</td>
<td>Advanced course, graphics programming</td>
</tr>
<tr>
<td>OGRE</td>
<td>Problem-based</td>
<td>Practical, hands-on learners</td>
<td>Object-oriented programming (OOP) teaching, intermediate to advanced courses</td>
</tr>
<tr>
<td>CryEngine</td>
<td>Project-based</td>
<td>Visual and kinesthetic learners</td>
<td>Advanced courses, high-fidelity game development</td>
</tr>
<tr>
<td>Irrlicht</td>
<td>Direct instruction</td>
<td>Systematic learners</td>
<td>Beginner to intermediate courses, low-poly game development</td>
</tr>
<tr>
<td>Unreal Engine</td>
<td>Demonstrative, project-based</td>
<td>Visual and kinesthetic learners</td>
<td>Intermediate to advanced courses, high-fidelity game development</td>
</tr>
<tr>
<td>Godot</td>
<td>Step-by-step, modular</td>
<td>Sequential learners</td>
<td>Beginner to advanced courses, 2D and 3D game development</td>
</tr>
<tr>
<td>Unity</td>
<td>Interactive, exploratory</td>
<td>Active, hands-on learners</td>
<td>Beginner to advanced courses, rapid prototyping, VR development</td>
</tr>
</tbody>
</table>

Table 1. A comparison of teaching and learning approaches in game engines.
4.1 Pedagogical value of game engines

Game engines transform the learning process into a more engaging and practical experience by offering students a platform to visualize and interact with abstract concepts. They provide a realistic context for students to grasp abstract concepts in game development and computer science, e.g., mirroring class objects with game objects or object manipulation through different components.

Game development calls for the application of problem-solving skills. Game engines challenge students to design, implement, and troubleshoot, reinforcing their theoretical knowledge with practical application. Game development offers students exposure to various fields, e.g., computer graphics, physics simulations, artificial intelligence, and more. In the case of Unreal Engine, students can experiment with AI behavior, graphical effects, or physics-based gameplay.

Game engines hold immense potential as pedagogical game development and computer science tools. As technology advances, they make learning an interactive, engaging experience. Educators can equip students with versatile skills beyond traditional computer science curricula.

4.2 Interactive tutorial in Unity Engine

For this tutorial, we decided to be inspired by the Unity Technologies company, which in 2017 decided to include an interactive tutorial directly in their game engine. The manual contained tasks in text form, updated after completion. It highlighted specific windows in the editor for the user to focus on. Unfortunately, due to the constant influx of new versions of Unity, this tutorial did not have time to adapt to the current versions, so standard tutorials replaced it.

Our goal was to create a manual that would eliminate the shortcomings of already existing manuals while preserving the parts that work. That is why we have set the following goals:

- Guiding users along a specific path using the level system, thanks to which we can refer to already-explained principles and thus avoid unnecessary repetition.
- Eliminating the monotony of regular instructions by adding multiple forms of explanation of functionalities in videos, text descriptions, and practical tasks.
- Enabling direct interaction with the environment by creating an integrated tutorial directly into Unity and adding tasks to practice the explained parts of the tutorial.
- Create a motivating, fun environment by adding various gamification elements, such as avatars, collecting money, and unlocking levels.
- Explanation of the fundamental Unity functionalities by creating six levels focused on the essential parts of the engine.

During the technical design of classes and components in Unity, we created the tutorial's basic structure as an easily scalable system of sections. Each section consists of subsections or steps: text, test, or video. Text steps create descriptions in text form, videos are linked to a video player, and tests are linked to scripts verifying student knowledge. Navigating the sections works by switching steps using the forward and backward buttons. The main manager is the manager who keeps the information about the current section and step, which is displayed and is responsible for switching steps.

The main idea of the tutorial was to create a comprehensive overview of Unity's basic functionalities, which will guide the user. Therefore, we designed six levels, divided into smaller parts dealing with specific Unity areas. The levels follow each other, while their last part follows the beginning of the next level. When creating the content, we tried to devote enough time to the structure of the sections, especially their alignment. It was necessary to find out which parts were related to each other to avoid confusion when going through the instructions.

We designed the environment to provide seamless navigation on screens and be user-friendly. The main screens include an introduction, the main menu, the shop, and a screen with instructions—text descriptions, video, and tasks.

The central part of the menu comprises the levels, which, when clicked, start a section with a given topic. They contain a button component that, when pressed, activates the corresponding section of the tutorial, and loads the last screen of the section. After going through the instructions, the locked next level will unlock, and the level will be set as completed.

We also created a store as a gamification element of motivation and reward, as users can unlock a new avatar for money earned by passing levels (Figure 12).

Steps are the smallest element in the tree of sections and thus are its leaves. There are three types of steps:

- **Text.** It is the simplest type of step, as it contains only a text string in the given step and a URL (uniform resource locator) link to the documentation. It is used for introductory descriptions of functionalities.

- **Video.** A video clip representing a detailed guide to the given part, playable within the tutorial.

- **Test/assignment.** It serves to verify students' knowledge through a short exercise. It contains a reference to the verification script, the text displayed at the beginning of the test and in the help, and a link to the video clip played in the help. The user must complete the task to get further in the tutorial.

The first part of the manual comprises texts introducing users to the specified issue's plot. The functionality is explained in more detail in the videos at each level. At most levels, practical tasks also verify and practice knowledge from a specific unit. The last level serves to review the acquired knowledge. Its content is a minigame (Figure 13) that contains intentional errors. To complete the level, the user must find and correct these errors, while the upper right corner of the screen shows how many errors he has already fixed.

The length of the instructions is interesting. While some students spent 10 h with it, others solved it in 1 h. Therefore, in the case of the Godot engine, we have followed the classic instructions.
4.3 Step-by-step tutorial for Godot engine

In the next part, we will describe a detailed guide to familiarize students with the basics of game development in the Godot engine. The goal was to provide a guide detailed enough to easily guide students through the essential elements of the game through practice. We wrote it so that it was obvious what was expected from the students and that they needed a minimum of assistance from the teacher. At the same time, we have included enough space for students’ creativity in the instructions.

Students have mastered a playable simple 2D platform game by fully completing the tutorial. They have a player with animations, a level composed of tiles, a background with a parallax effect, a functional collision system, an interactive object,
a moving camera, or gravity. The instructions\textsuperscript{29} are divided into the fundamental part, where the students are fully guided, and the supplementary part, which consists of additional tasks. The idea was to explain and let the students practically try out all the basic functionalities of the Godot engine so that they could playfully continue their own in the additional part. And if they start additional tasks, the number of collected objects and the number of lives that can be lost or gained in diverse ways are recorded in the user interface (UI), and there is also a way to win the game (Figure 14).

Although the instructions are long, the students were fine with it. Compared to Unity’s interactive tutorial, the solution time has stabilized at 3–4 h. That is why we have returned to the standard tutorial\textsuperscript{30} for the Unreal engine.

### 4.4 Step-by-step tutorial for unreal engine

The difference between a step-by-step and an interactive tutorial is that the individual steps are not described inside the engine environment but in a separate tutorial. The difference between the step-by-step instructions for the Godot engine and UE was that while with Godot, we got by with written instructions with a minimum of figures, with Unreal, it was easier to use figures due to the structure of the environment (Figure 15).

The effect was as expected; the students spent a similar amount of time, 3–4 h, solving the tasks.

---


4.5 Impact on learning outcomes

We have mentioned that game engines offer various features suitable for teaching styles and learning preferences. Now, we will discuss how these differences in features impact learning outcomes. The effects of these differences depend on several factors, like individual learning styles, project nature, or instructional design. For instance, game engines such as Unity, with their user-friendly interface and intuitive design, may reduce cognitive load for beginners, enabling them to grasp fundamental concepts more quickly. This reduced cognitive load can provide an environment where learners can experiment freely and learn by doing, leading to increased engagement and improved understanding of core game development concepts.

On the other hand, more complex game engines, like Unreal Engine, offer advanced functionalities that can provide opportunities for deep learning and mastery for more advanced learners. The complexity of these engines could promote a deeper understanding of the intricate mechanics of game development, enhancing problem-solving and critical-thinking skills. Table 2 showcases the impact of the three different tutorial approaches on various learning outcomes, highlighting their unique strengths and features. For instance, Unity’s interactive tutorial stands out for its emphasis on interactive exercises and gamification, while the step-by-step tutorials for Godot and Unreal Engine encourage independent learning and reinforcement through repetition. Visual learning support is a unique strength of the Unreal Engine.

5. Summary and annual game jam experience

Since games intersect many disciplines, including art, sound, storytelling, or design, the game engines allow the introduction of these elements next to the more technical aspects of computer science. The tutorial’s diversity demonstrates game engines’ adaptability to different learning styles and underscores their value as practical teaching tools. Moreover, Unity and Unreal engines are known for their comprehensive VR support, offering various built-in features.
The Role of Game Engines in Game Development and Teaching
DOI: http://dx.doi.org/10.5772/intechopen.1002257

A highlight of our exploration is the hands-on approach to learning, represented by interactive and step-by-step tutorials in game engines. Each tutorial approach offers unique insights. The Unity tutorial adopted an interactive learning approach, reinforcing theoretical concepts with practical application and fostering a dynamic understanding of game development in Unity. Such a tutorial suits learners who prefer active learning and are comfortable with some degree of self-direction. Given its interactive nature, the time spent on tasks can vary among students, ranging from 1 to 10 h. This variability reflects the tutorial’s exploratory nature and the learners’ potential struggles within the engine environment.

On the other hand, the Godot tutorial, a step-by-step guide, allowed readers to build their understanding progressively, each step building upon the last to create a complete picture of game development in Godot. Such a tutorial is ideal for a beginner or those who prefer a structured approach to learning. With clear instructions, students spend a stable amount of time on tasks, approx. 3–4 h. This stability indicates the straightforward, well-defined nature of the tutorial and students’ habit of working with structured documentation. In our case, structured materials are a natural part of institutional courses.

Compared with the Godot tutorial, the Unreal Engine tutorial focused on a step-by-step process combined with visual aids. However, instead of programming the scripts, we used blueprints. Such figure-based learning benefits visual learners, enabling a clear understanding of complex processes involved in game development using the engine. Like the Godot tutorial, the time commitment for tasks is stable, with students spending approx. 3–4 h. The consistency in time spent suggests that learners can expect a steady pace of progress.

From 2018 to 2020, our curriculum focused more broadly on game design and programming technologies. However, reflecting increased student interest and the industry’s evolving needs, we began integrating specific game engines into our curriculum in 2021, as shown in Table 3. By 2023, we had successfully incorporated the Unity, Godot, and Unreal Engine into our syllabus.

Table 4 shows our students’ changes in technology preferences during our annual game jam events, alongside participation numbers. Unity was the dominant choice

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Engine understanding</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Active learning</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Self-paced learning</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Problem-solving skills</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Feedback and reflection</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Interactive exercises</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Motivation through gamification</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Reinforcement through repetition</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Independent learning</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Visual learning support</td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

Table 2. Tutorials’ impact on learning outcomes.

A highlight of our exploration is the hands-on approach to learning, represented by interactive and step-by-step tutorials in game engines. Each tutorial approach offers unique insights. The Unity tutorial adopted an interactive learning approach, reinforcing theoretical concepts with practical application and fostering a dynamic understanding of game development in Unity. Such a tutorial suits learners who prefer active learning and are comfortable with some degree of self-direction. Given its interactive nature, the time spent on tasks can vary among students, ranging from 1 to 10 h. This variability reflects the tutorial’s exploratory nature and the learners’ potential struggles within the engine environment.

On the other hand, the Godot tutorial, a step-by-step guide, allowed readers to build their understanding progressively, each step building upon the last to create a complete picture of game development in Godot. Such a tutorial is ideal for a beginner or those who prefer a structured approach to learning. With clear instructions, students spend a stable amount of time on tasks, approx. 3–4 h. This stability indicates the straightforward, well-defined nature of the tutorial and students’ habit of working with structured documentation. In our case, structured materials are a natural part of institutional courses.

Compared with the Godot tutorial, the Unreal Engine tutorial focused on a step-by-step process combined with visual aids. However, instead of programming the scripts, we used blueprints. Such figure-based learning benefits visual learners, enabling a clear understanding of complex processes involved in game development using the engine. Like the Godot tutorial, the time commitment for tasks is stable, with students spending approx. 3–4 h. The consistency in time spent suggests that learners can expect a steady pace of progress.

From 2018 to 2020, our curriculum focused more broadly on game design and programming technologies. However, reflecting increased student interest and the industry’s evolving needs, we began integrating specific game engines into our curriculum in 2021, as shown in Table 3. By 2023, we had successfully incorporated the Unity, Godot, and Unreal Engine into our syllabus.

Table 4 shows our students’ changes in technology preferences during our annual game jam events, alongside participation numbers. Unity was the dominant choice
Computer Science for Game Development and Game Development for Computer Science

In 2020, but with the introduction of Godot into our curriculum in 2021, we saw a shift in preference. By 2023\(^\text{31}\), we saw a more balanced distribution among the three engines, demonstrating that our students acquired diverse skills. Of note, the total number of participants and teams substantially increased in 2023, highlighting the growing interest in our game design and development courses.

As Table 5 illustrates, incorporating game engines into our curriculum has marked improvements in our students’ experience with the engines. Based on a questionnaire, most students reported having no or low skill levels at the beginning of the semester. However, by the end of the semester, a significant shift toward moderate and high skill levels is observable. The data suggest our curriculum’s focus on hands-on

\(^{31}\) Games of 2023’s Game Jam: [cit. 2023-06]. URL: https://itch.io/jam/game-jam-kosice-2023
learning with various game engines equips our students with applicable skills in game
development, preparing them for diverse opportunities in the industry.

These results indicate the effectiveness of different game engines in teaching
game development, each offering unique strengths to meet diverse learner needs and
project requirements.

6. Conclusion

In this chapter, we have explored the role of game engines in game develop-
ment and learning. Through numerous examples, we have illustrated how these
ingines employ their powerful capabilities to craft immersive digital experiences.
Importantly, we have highlighted how game engines can extend beyond creation into
education as practical game design tools.

In our department, we focus on education in software engineering. So, in the
design and development of games, the implementation side, including the design
process (in this sense, also the use of game engines), is vital to us. We are looking for
ways and activities to teach students to transfer formalisms and critical aspects of
Game Theory into Computer science practice. It is not only a transfer into source code
but also a design or feedback.

The choice of an appropriate game engine significantly impacts a game's perfor-
ance, capabilities, and overall development process. When considering the educational
impact, tutorials in these engines give students a practical understanding of game
design and development, equipping them with skills highly demanded in the industry.
Our insights from exploring interactive and step-by-step tutorials in Unity, Godot, and
Unreal Engine further emphasized this point. The variability in the time spent on tasks
across these tutorials reflected their adaptability to different learning styles.

As demonstrated in our curriculum evolution and students' improved experience
in game engine technologies, our teaching approach empowers students with the
critical skills needed. Furthermore, it fosters a dynamic learning environment full
of motivation and creativity. As we adapt to technological advances and align with
industry trends, we envision a responsive, anticipative, and nurturing future of game
design education.

Moreover, our annual game jam event revealed these engines' real-world relevance
and application in a learning context. The engine preferences of our students under-
scored the need for a diverse toolkit in game development education. Integrating
game engines into the curriculum can provide more engaging and immersive learning
experiences, revolutionizing the way we teach game design and computer science.

Understanding and exploring game engines is crucial for anyone interested in game
development. It offers a blend of creativity and technical skills, fostering a comprehen-
sive approach to learning. We anticipate further research and exploration in this exciting
field, keeping up with the continuous evolution of game development technologies.

Acknowledgements

This work has been supported by Slovak KEA Agency under Grant No.
048TUKE-4/2022: “Collaborative virtual reality technologies in the educational

Author details

Branislav Sobota* and Emília Pietriková
Technical University of Košice, Slovakia

*Address all correspondence to: branislav.sobota@tuke.sk

© 2023 The Author(s). Licensee IntechOpen. This chapter is distributed under the terms of the Creative Commons Attribution License (http://creativecommons.org/licenses/by/3.0), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.
References


[16] Gestwicki P. Godot engine and checklist-based specifications: Revising a game programming class for asynchronous online teaching. Journal
of Computing Sciences in Colleges.