Game engine impact on hardware resources

Mariia Yu. Tiahunova, Halyna H. Kyrychek, Alina Kazurova, Yehor Knyryk and Dmytro Tiahunov

National University "Zaporizhzhia Polytechnic", 64 Universytetska St., Zaporizhzhia, 69063, Ukraine

Abstract

The article reviews and analyzes the qualitative characteristics of several game engines that are widely used today, including Unity, Unreal Engine, Godot Engine, and Redot Engine. For the purpose of game development, Unreal Engine 5.4.4 was selected in combination with the Blueprint visual programming language, owing to its advanced rendering capabilities, flexibility, and widespread adoption in contemporary game design. The study outlines the general concept and key structural components of the project, such as genre, target audience, central idea, main character, setting, objectives, target platform, operating system support, difficulty level, and gameplay duration. The main aim of the research is the development of a computer game in the 3D platformer genre and the systematic investigation of the influence exerted by the selected game engine on computer hardware resources. The research object is the process of game implementation and a comparative evaluation of its performance across different engines. The research subject is the performance indicators of a computer game realized on various game engines, which include frame rate (frames per second), GPU and CPU utilization, as well as the effect of graphical settings on these metrics. To achieve the stated objectives, the following tasks have been defined: to analyze and compare existing models, technologies, and state-of-the-art game engines; to formulate the game concept; to implement the project architecture and user interface, along with game objects, characters, and environments; to conduct an experimental assessment of the impact of the game engine on hardware resources; and to provide a detailed analysis and interpretation of the obtained results in the context of contemporary trends in computer game development.

Keywords

Blueprint, Unity, Unreal Engine, game, computer, visualization, interface, performance

1. Introduction

In recent years, the development and implementation of computer games has emerged as a rapidly expanding field, which, owing to the continuous integration of technological innovations, has transformed into a global industry with multi-billion-dollars revenues [1]. At present, games serve not only as a form of entertainment but also as a valuable tool in education, as they foster the development of cognitive skills, teamwork, and critical thinking [2]. Computer games showcase modern data processing approaches and support the optimization of web systems through technologies and methods [3]. Moreover, modern 3D platformers such as Super Mario Odyssey, A Hat in Time, and Psychonauts 2 illustrate the high popularity of this genre [4, 5]. Players particularly value the freedom of exploration, innovative mechanics, and adventure elements that these games provide [6]. The 3D platformer genre offers broad creative potential, as it allows for the integration of unique mechanics, open-world environments, and complex level designs [7]. This versatility enables developers to design games characterized by a high degree of interactivity and aesthetic quality. Game engines play a crucial role in this process, offering advanced graphics capabilities and user-friendly tools for 3D game development. Such engines empower even small development teams to create projects with a high level of detail and sophistication.

CS&SE@SW 2025: 8th Workshop for Young Scientists in Computer Science & Software Engineering, November 21, 2025, Kryvyi Rih, Ukraine
https://doi.org/10.55056/cssesw2025/paper08
mary.tyagunova@gmail.com (M. Yu. Tiahunova); kirgal08@gmail.com (H. H. Kyrychek); akazurova@gmail.com (A. Kazurova); egorknirik2@gmail.com (Y. Knyryk); d.tiahunov@gmail.com (D. Tiahunov)
https://zp.edu.ua/worker/tiahunova-mariia-iuriivna/ (M. Yu. Tiahunova);
https://zp.edu.ua/worker/kyrychek-halyna-hryhorivna/ (H. H. Kyrychek)
0000-0002-9166-5897 (M. Yu. Tiahunova); 0000-0002-0405-7122 (H. H. Kyrychek); 0000-0001-7382-5204 (A. Kazurova); 0009-0007-8273-5797 (Y. Knyryk); 0009-0009-8345-5089 (D. Tiahunov)

Consequently, the research topic addressed in this study is of significant relevance.

The primary *aim* of the research is the development of a computer game in the 3D platformer genre and the systematic investigation of the influence exerted by the selected game engine on computer hardware resources.

The *object of the research* is the process of game implementation and the comparative evaluation of its performance across different engines.

The *subject of the research* is the performance indicators of a computer game realized on various game engines, which include frame rate (frames per second), GPU and CPU utilization, as well as the effect of graphical settings on these metrics.

To achieve the stated objectives, the following *tasks* have been defined: to analyze and compare existing models, technologies, and state-of-the-art game engines; to formulate the game concept; to implement the project architecture and user interface, along with game objects, characters, and environments; to conduct an experimental assessment of the impact of the game engine on hardware resources; and to provide a detailed analysis and interpretation of the obtained results in the context of contemporary trends in computer game development.

There are several parameters by which computer games can be classified: by game genre, by monetization system, by the number of players involved, by platform, and by the age categories of players. A genre of game represents a set of games united by common characteristics of game-play. Typically, the primary criterion for defining a genre is the type of action performed by the player. A wide range of genres has been identified in modern game studies. For instance:

- platformer – movement of a character across various platforms while overcoming obstacles [8];
- horror – creation of a tense and immersive atmosphere [9];
- fighting – individual or team-based combat scenarios [10];
- survival – maintaining life in a hostile or dangerous environment [11];
- interactive movie – providing players with branching narrative choices at key moments, thereby influencing the storyline [12];
- rhythm games – performing actions in synchronization with musical patterns [10];
- sandbox – free interaction with an open world under self-defined rules [13];
- educational games – focusing on diverse topics and skill development [14].

In addition to genre-based categorization, computer games are also classified according to the age appropriateness of players, which is regulated through specialized rating systems. Among the most widely recognized are the Pan European Game Information (PEGI), the Entertainment Software Rating Board (ESRB), the Unterhaltungssoftware Selbstkontrolle (USK), and the Computer Entertainment Rating Organization (CERO) [15].

2. Project structure

The development of computer games is a structured and well-defined process that involves specific stages. However, the release of certain games may take several years, and even after a game is launched, its modules and functional processes often require refinement and further development. A game engine is specialized software that enables developers to efficiently create and manage various aspects of a game by integrating libraries and tools that accelerate development, simplify visualization, scripting, sound management, and other processes. In the present research, an analysis of the most commonly used game engines is conducted:

- Unity – supports projects of varying complexity, is used across multiple industries, and offers intuitive tools that streamline the development process [16];
- Unreal Engine (UE) – renowned for its realistic rendering capabilities, tools for creating large open worlds, and cinematic effect features [17];

- Godot Engine – designed for both 2D and 3D game development, popular among indie developers, and widely used for educational purposes and prototyping [18];
- Redot Engine – a multifunctional engine designed for developing both 2D and 3D games through a unified interface that provides a comprehensive set of tools [19].

The advantages of Unity include: support for modern platforms; user-friendly interface; an active community and abundant learning resources; tools for VR and AR development; and the Bolt visual scripting language. However, disadvantages include performance issues in large-scale projects [20]. Advantages of Unreal Engine include: high-quality graphics; a large developer community; extensive learning materials; open-source engine code; the visual scripting language Blueprint; and powerful development tools. Disadvantages include: licensing costs; high system requirements; a steep learning curve; and a complex interface [17]. The advantages of Godot Engine include: free and open-source nature; robust 2D capabilities; a simple and flexible node-based architecture; and support for GDScript and other programming languages. There are disadvantages: limited educational resources for specific tasks; the need for configuration and optimization; and a restricted number of third-party tools and resources [18]. A comparative analysis of game engines is presented in table 1.

Table 1
Comparison of game engines.

Criterion	Unity	Unreal Engine	Godot Engine	Redot Engine
Learning difficulty	Low	High	Average	Average
Multiplatform	+	+	+	+
Graphics level	Average	High	Average	Average
Community	High	High	Average	Low
Language	Bolt, C#	C++, Blueprint	GDScript, C, C++, C#	GDScript, C, C++, C#

By comparing the characteristics and analyzing the aforementioned game engines, Unreal Engine was selected for the development of the game, including its core modules and objects. Furthermore, after comparing key features and analyzing the functional capabilities of programming languages, the decision was made to adopt Blueprint. Blueprint is a visual scripting system integrated into Unreal Engine that enables the creation of game logic without traditional coding. Through the connection of nodes within a graphical interface, Blueprint allows the implementation of complex functions and interactions.

The comparative analysis of game engine performance was conducted using the game The Ball, implemented on both Unity 2021.3.15f1 and Unreal Engine 5.4.4. Performance testing was performed under controlled conditions to ensure consistency and reliability of results. The chosen test environment consisted of a computer system with the following specifications: Ryzen 5 5500 processor, 16 GB of RAM, GTX 1050 TI graphics card, 256 GB SSD, and Windows 11 Pro 23H2 operating system. During testing, key performance metrics were recorded, including frame rate (frames per second), CPU and GPU load, and system resource usage under varying graphical settings. The objective of this methodology was to obtain empirical data on the impact of the selected game engine on hardware performance, thereby enabling a scientifically grounded comparison between the two engines.

The parameters of the game The Ball are as follows:

- game genre: 3D platformer;
- target audience: children aged 5 to 12 years;
- vCore concept: the player controls a game character moving across platforms from the beginning to the end of the game zone while overcoming various obstacles;
- character: the game character is a football (soccer) ball controlled by the player;
- setting: the game action takes place high in the sky on floating platforms;
- objective: to complete all levels;

- platform: the game is designed for personal computers (laptops);
- operating system support: compatible with Windows operating systems starting from Windows 10;
- difficulty level: low overall, however, depending on the player's age and skill in interacting with a mouse and keyboard, the game can become considerably challenging;
- game duration: completing the entire game takes approximately 30 minutes; however, the duration depends on the player's style, completing a single level takes about 3 minutes.

The game The Ball employs a variety of gameplay mechanics designed to enhance interactivity and challenge within the 3D platformer genre. These mechanics can be categorized into platform mechanics, object mechanics, and character mechanics, each contributing to the overall gameplay experience. Platform mechanics define the behavior of the game's structural elements, which the player must navigate to progress:

- platform fall mechanic: upon contact with a playable character, the platform begins to collapse after a short delay and is subsequently removed from the level, preventing reuse, as well as this mechanic introduces time-based challenges and requires player precision;
- platform movement mechanic: platforms traverse between two predefined points, pausing briefly upon reaching each endpoint, as well as this mechanic requires the player to anticipate movement timing and plan accordingly;
- platform rotation mechanic: platforms rotate 360 degrees and pause momentarily, presenting dynamic navigation challenges that require spatial awareness and timing.

Game objects are interactive elements that influence player progression and gameplay dynamics:

- chest mechanic: a chest that must be manipulated by the playable character to progress, as well as this mechanic encourages problem-solving and interaction;
- spikes mechanic: a stationary hazard object which destroys the player character upon contact, resulting in level restart, adds an element of risk and requires careful navigation;
- teleport mechanic: enables instantaneous transportation of the player character to a different section of the level, facilitating dynamic exploration and level design complexity.

Character mechanics define the movement and interaction capabilities of the player-controlled entity:

- movement mechanic: the character moves forward, backward, left, and right based on physical input, movement dynamics are influenced by the camera angle, adding complexity to navigation;
- camera control mechanic: the camera system maintains focus on the character, allowing rotational control for exploration of the environment, zoom functions are included, with constraints to ensure gameplay balance;
- character destruction mechanic: collision with spikes results in the disintegration of the character into multiple spheres, visually reinforcing failure and adding immersive feedback.

This structured approach to game mechanics not only enhances the player's experience but also contributes to the educational and experimental value of the game development process, particularly in the context of evaluating engine performance and hardware resource usage. A game loss can occur in three cases: the character falling beyond the boundaries of the level; the player character coming into contact with spikes; or the accidental activation of a falling platform, resulting in the inability to navigate the game environment. In the first two cases, the level is automatically restarted. In the third case, the level must be restarted manually via the menu.

3. Implementation

The process of creating a prototype represents a critical stage in game development, serving as a preliminary implementation of a simplified model to verify design concepts, test core functionalities, and demonstrate both gameplay ideas and user interface structures. Prototyping enables iterative testing and refinement before full-scale production, thereby reducing development risks and improving efficiency. Prototypes can be categorized into low-fidelity and high-fidelity models. Low-fidelity prototypes represent schematic sketches or simplified mockups designed to provide a general understanding of the game structure and interactions. These prototypes focus on conceptual clarity without detailed graphical representation. They are particularly useful in the early stages of development for brainstorming and evaluating gameplay ideas. High-fidelity prototypes are detailed models that incorporate graphic design, animation, and interactive elements closely resembling the final version of the game. These prototypes allow for comprehensive usability testing and realistic feedback on game mechanics and interface design.

For example, a high-fidelity prototype of the game settings menu was developed to test user interaction and functionality prior to the final implementation. This prototype, illustrated in figure 1, includes interactive buttons, adjustable settings, and a user-friendly interface layout, reflecting both aesthetic and functional design considerations. The prototyping stage is essential not only for design verification but also for aligning the development process with project objectives, ensuring the final product meets both user expectations and technical requirements.

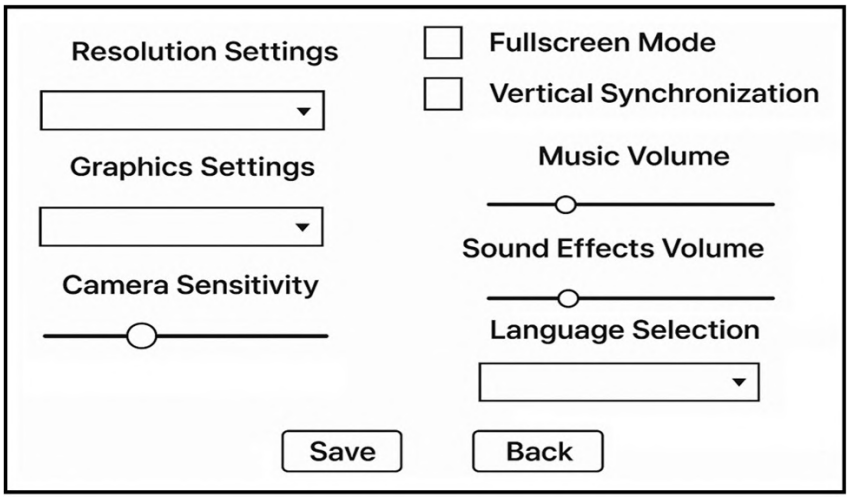


Figure 1: Implementation of the pause menu functionality.

The game interface was implemented using Widget Blueprint, utilizing a set of basic widgets:

- canvas panel – the primary container for creating the user interface;
- widget switcher – allows switching between different child widgets;
- size box – enables control over the size of child widgets;
- vertical box – allows arranging child widgets vertically in a column;
- horizontal box – allows arranging child widgets horizontally in a row;
- scroll box – allows displaying a large number of widgets within a limited area with scrolling functionality;
- border – provides a background frame around a child widget;
- combo box – allows the user to select one item from a dropdown list;
- check box – enables users to select or deselect an option;
- button – an interactive widget that responds to user clicks;
- text – displays textual information in the user interface;

- image – displays images within the user interface;
- spacer – used to add spacing between widgets;
- slider – allows the user to select a value from a specified range using a sliding control.

A separate level was created for the main menu. Camera rotation within this level is implemented using a specialized script. The level blueprint is responsible for creating the user interface and generating a save file for game progress. Additionally, the following functionalities were implemented: a game instance responsible for creating a save file for sound and mouse sensitivity settings when the game is first launched; the main menu, tutorial windows, and character control information windows; the main menu buttons and level selection menu; the level unlock system, game settings menu, getting current settings, and functionality for changing game settings; and an algorithm supporting pause functionality through a special pause menu (figure 2).

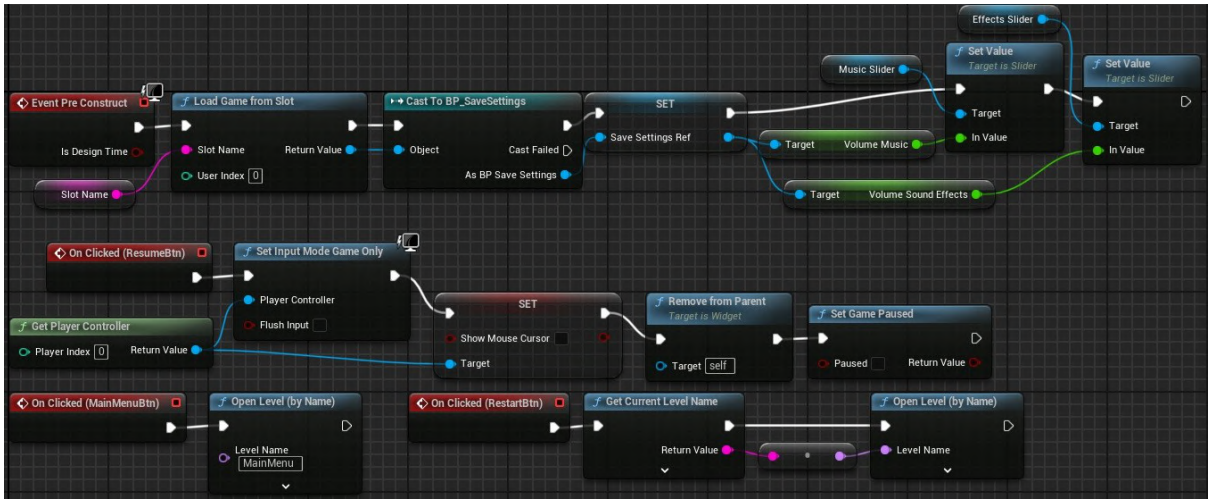


Figure 2: Implementation of the pause menu functionality.

Game objects are created based on the actor class. Several components are used in their implementation:

- static mesh – represents a 3D model, what is used for constructing levels in Unreal Engine, are created in specialized modeling software and then imported into the game engine, and levels generally consist of actors with a static mesh component;
- camera – an object that defines the point of view from which the player observes the virtual game world;
- spring arm – a component that maintains child elements at a fixed distance from their parent object and then upon collision, it moves the child elements closer to the parent and returns them when collisions cease;
- box collision – used for detecting collisions or interactions between objects in the game, is a rectangular or cubic area that checks for collisions, allowing detection of when an object enters or leaves the defined area;
- sphere collision – functionally similar to the box collision component but shaped as a sphere.

Each game object is implemented with its own logic, including both its visual representation and functional properties. For example, a movable platform object has its own appearance and includes the implementation of platform movement and fall mechanics. The platform fall mechanic is implemented based on a principle that checks whether an object has entered the platform's collision area. If this object is the game character, a delay of 1–2 seconds is triggered, after which physics simulation is enabled and the platform begins to fall. After falling for 5 seconds, the platform is removed. The rotating

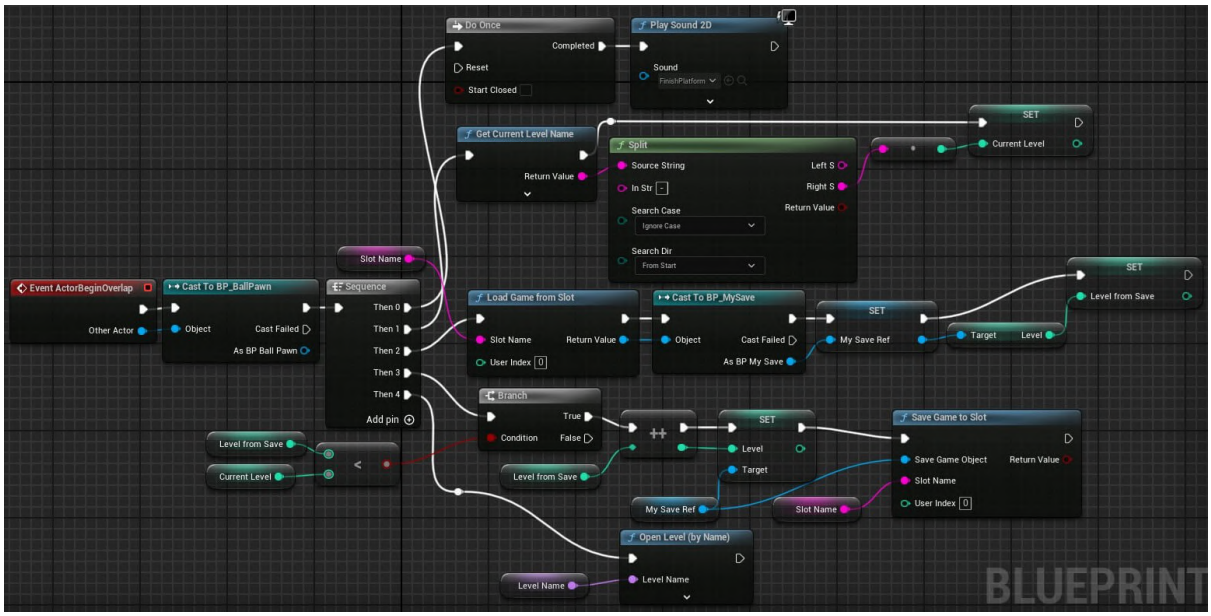


Figure 3: Implementation of the finishing platform logic.

platform object, besides its visual representation, includes an implementation of the platform rotation mechanic. Figure 3 illustrates the logic implementation of the game object “Finish Platform”.

The game object “Chest” possesses not only a unique visual design but also a specific script implementation for its removal when falling beyond the boundaries of the game area. The operation principle is based on periodically checking the height coordinate of the object. If the height coordinate exceeds a predefined threshold value, the object is removed from the level. This game object also simulates physics to allow the player character to push it, adding interactive depth to the gameplay. The game object “Spikes” similarly features a distinct visual design and implements its own mechanics. Upon contact with the player character, these spikes trigger the destruction of the character and reset the level. Figure 4 illustrates all game objects utilized in level construction. Basic platforms lack game logic and serve solely as surfaces for the player character to traverse. These platforms are colored green and vary only in size. The starting platform, while also categorized as a basic platform, differs by having a yellow color to visually denote its function.

Figure 4: Game objects for level construction.

The player character was created using the pawn class, with the addition of the following components: static mesh, camera, and spring arm. The visual appearance of the player character is illustrated in figure 5.

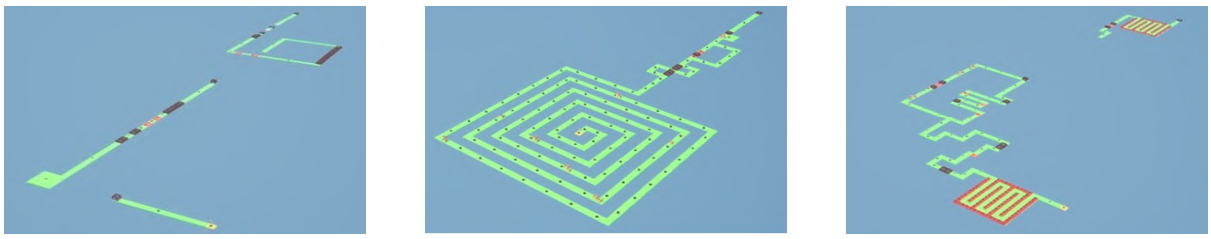
Control axes for the player character were defined to manage movement and camera operation. These

Figure 5: Implementation of the pause menu functionality.

axes were configured in the project settings. Character movement was implemented using physics simulation; therefore, physics simulation was enabled for the static mesh component, and its parameters were appropriately configured.

Additionally, camera control mechanics were implemented, including zooming in and out relative to the player character. Given the possibility that the character may fall off platforms during gameplay, a script was implemented to monitor the character's height coordinate. If the character falls below a predetermined threshold, the level is automatically restarted. Furthermore, a mechanic for the character's disintegration was implemented, where the character breaks into several spheres that are removed after a specified time interval. Using the created game objects, ten game locations were constructed. Each game locations implemented as a separate scene. Some examples are as follows:

- game location (1) introduces the player to the primary game objects: falling platforms, rotating platforms, teleporters, and moving platforms;
- game location (2) familiarizes the player with the game object "Chest" and teaches interaction with it;
- game location (3) introduces the player to the game object "Spikes" (figure 6);
- game location (4) is a labyrinth constructed with falling platforms: teleporters are located at the edges of the labyrinth, leading to distant platforms, and only one of four teleporters leads to the finish platform;
- game location (6) is a teleport labyrinth, where the player must choose the correct path;
- game location (9) represents a classic labyrinth structure (figure 7).

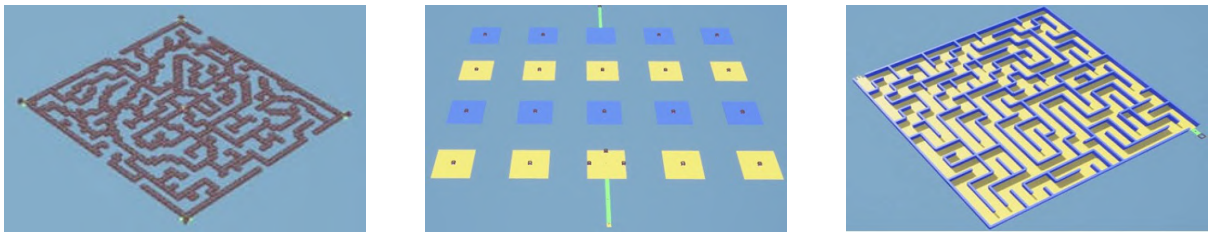


(a) (b) (c)

Figure 6: Game locations (a) №1, (b) №2 and (c) №3.

To compile the project in Unreal Engine, specialized tools were utilized. For a project supporting the Windows operating system, the target platform and build configuration were selected. Four build configuration options are available: DebugGame, Development, Shipping, and Use Project Setting (Shipping).

The first two configurations generate additional files that are not required for final distribution. The Shipping configuration, in contrast, does not produce files necessary for debugging and testing the



(a) (b) (c)

Figure 7: Game locations (a) №4, (b) №6 and (c) №9.

game. This configuration is used to achieve optimal performance and deployment of the game. It removes console commands, statistical tools, and profiling utilities, resulting in a clean build containing only the essential files required for the game's operation and execution.

4. Results and research

An installer simplifies the process of installing software on users' computers. Installers provide several benefits, including: creation of shortcuts, adding the program to the list of installed applications, installation of additional components, modification of installation parameters, and reduction of software size using compression algorithms to ensure faster downloading over a network. Inno Setup is one of the most popular programs for creating installers. It allows the creation of installers for Windows operating systems either through scripting or using a wizard interface. An installer for Windows was created using inno setup.

The installer can be viewed and downloaded via the link available on the GitHub page [21]. Similarly, the version of the game developed in Unity 2021.3.15f1 can also be viewed and downloaded via the link on the GitHub page [22]. This version of the game is used for comparative analysis. A general description of the development process of the game in Unity 2021.3.15f1 is provided via the link [23]. The next step involved conducting a monitoring of computer resource usage under different graphics settings in the game. Monitoring was performed with vertical synchronization (V-Sync) disabled and a screen resolution set to 1920 × 1080. For the purpose of research, comparison, and result analysis, the game implemented in Unity 2021.3.15f1 was utilized. Monitoring computer resources allows for tracking the status of key system components — including the CPU, RAM, storage devices, and network — to analyze performance, prevent overload, and identify potential issues. Specialized software tools, MSI Afterburner and RivaTuner Statistics Server (RTSS), were used for resource monitoring.

MSI Afterburner is a free software solution for overclocking graphics cards and hardware monitoring of computers. It supports not only MSI products but also graphics cards from other manufacturers.

RivaTuner Statistics Server (RTSS) is a utility for real-time hardware monitoring. It typically displays frame rate, hardware sensor data, and other performance indicators in games and applications. RTSS can receive and display data from programs such as AIDA64, MSI Afterburner, and HWiNFO. The parameters displayed on the screen are selected in the Monitoring Settings section of MSI Afterburner. Once MSI Afterburner and RivaTuner Statistics Server are configured, launching the game will display the chosen monitoring indicators in real time. Figure 8 illustrates the operation of the configured monitoring system.

The computer used to run the game is equipped with the Windows 11 23H2 operating system. For testing purposes, the system is configured with the following specifications: Game Ready driver version 566.14 (released Tue Nov 12, 2024); AMD Ryzen 5 5500 processor; 16 GB RAM; NVIDIA GeForce GTX 1050 Ti graphics card; storage devices including an HDD with 931.5 GB and an SSD with 232.9 GB capacity. The results of the resource monitoring for the game implemented on Unreal Engine 5.4.4 are presented in figure 9.

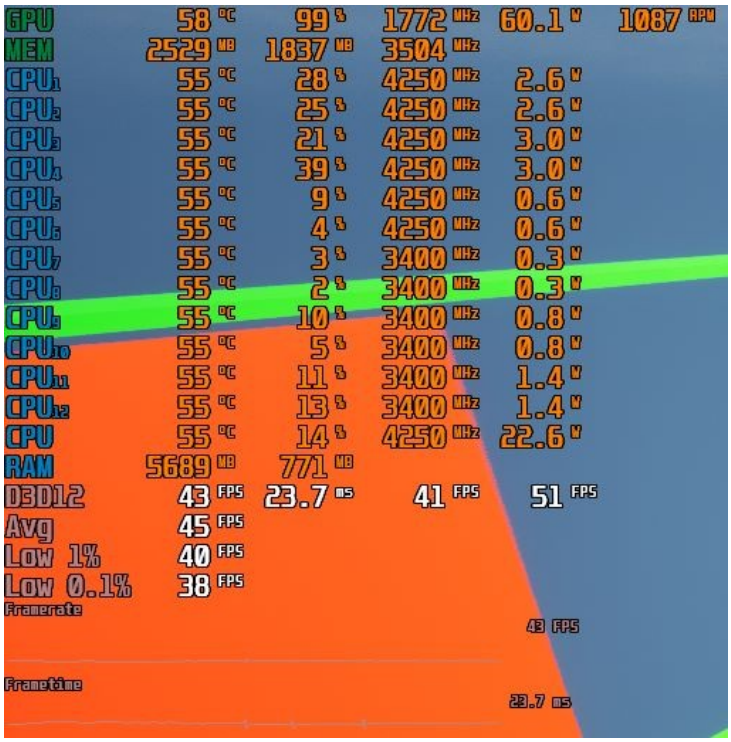
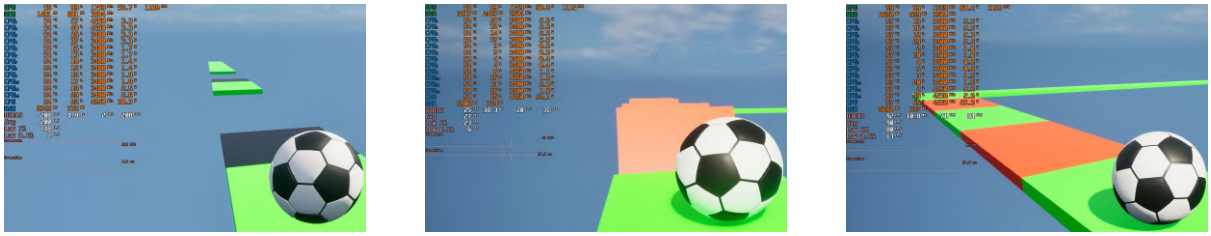


Figure 8: Configured monitoring system.



(a) (b) (c)

Figure 9: a) at low graphics settings, b) at medium graphics settings, c) at maximum graphics settings.

And the results of monitoring the game based on Unity 2021.3.15f1 are shown in figure 10.

The obtained results were analyzed by comparing the impact of various parameters on hardware resources. Considering that the game file size is an important metric, it should be noted that the file size of the game developed in Unreal Engine is 405 MB, which is 312 MB larger than the game developed in Unity, which has a file size of 93 MB. The comparison chart of the average frame rate at low, medium, and maximum graphics settings is presented in figure 11.

The chart shows that the game in Unreal Engine (at low settings) displays an average of 1574 frames per second fewer than the game in Unity. It is important to note that a smooth gaming experience typically requires between 30 and 60 frames per second. At medium settings, Unreal Engine renders an average of 1106 frames per second fewer than Unity. At maximum settings, Unreal Engine renders on average 624 frames per second fewer than Unity.

The comparison chart of CPU load at low, medium, and maximum graphics settings is presented in figure 12.

The chart shows that the game in Unreal Engine (at low settings) loads the CPU by 3% more than the game in Unity. At medium settings, the Unreal Engine game loads the CPU by 2% more than Unity. At maximum settings, the CPU load in Unreal Engine is 7% higher compared to Unity.

(a) (b) (c)

Figure 10: a) at low graphics settings, b) at medium graphics settings, c) at maximum graphics settings.

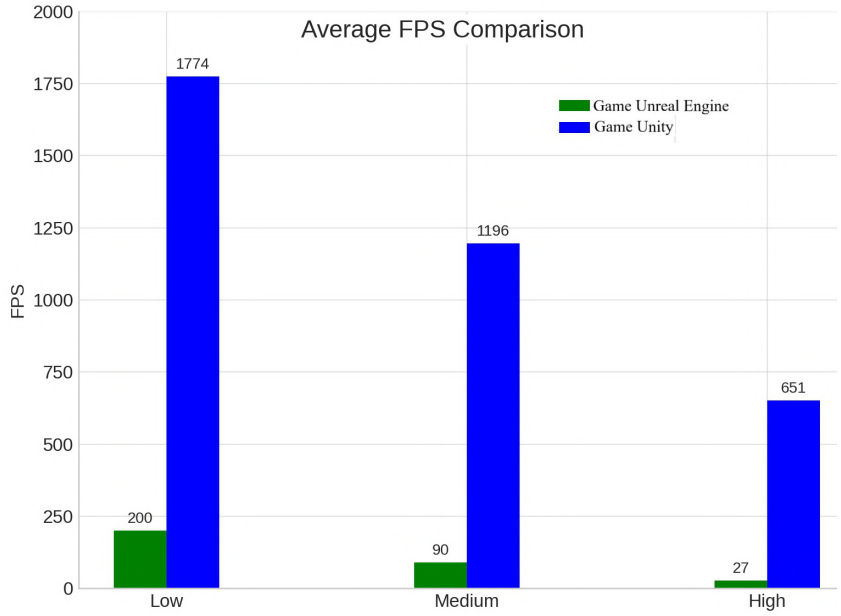


Figure 11: Average frames per second at low, medium, and maximum graphics settings.

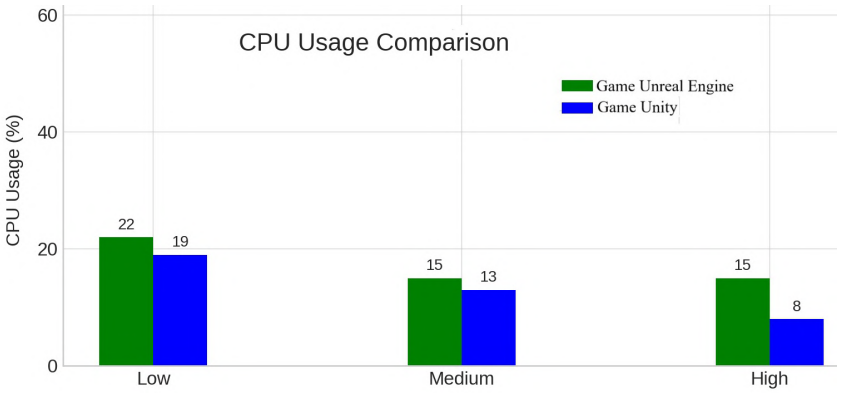


Figure 12: CPU load at low, medium, and maximum graphics settings.

The comparison chart of GPU load at low, medium, and maximum graphics settings is presented in figure 13.

The chart shows that the game in Unreal Engine (at low settings) loads the GPU by 25% more than the game in Unity. At medium settings, the Unreal Engine game loads the GPU by 12% more than Unity. At maximum settings, the GPU load in Unreal Engine is only 4% higher compared to Unity.

The comparison chart of video memory usage by the game at low, medium, and maximum graphics

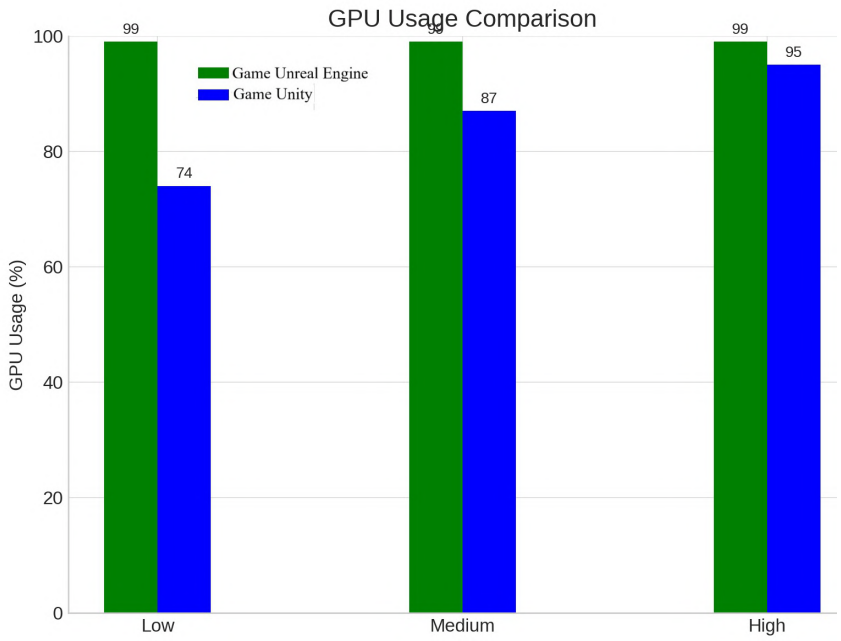


Figure 13: GPU load at low, medium, and maximum graphics settings.

settings is presented in figure 14.

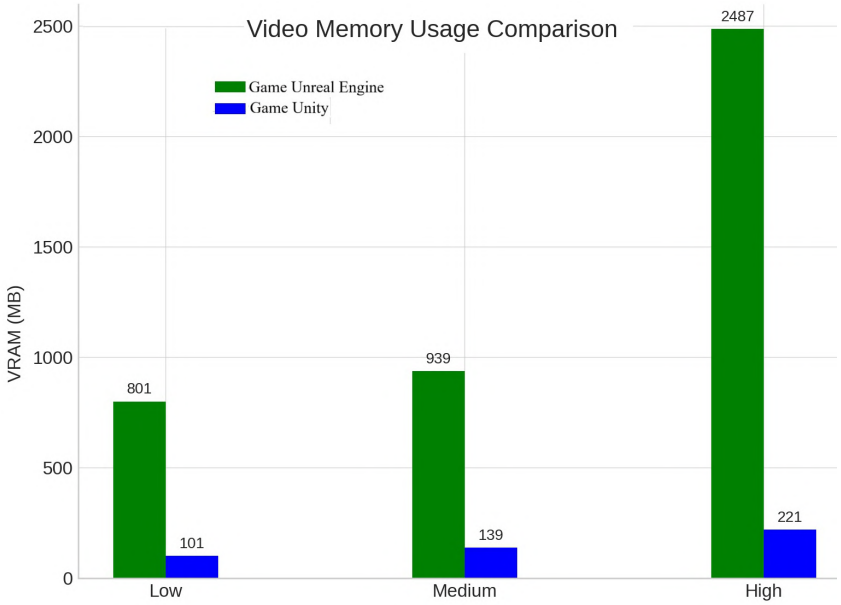


Figure 14: Game's video memory usage at low, medium, and maximum game graphics settings.

The chart shows that the game in Unreal Engine (at low settings) uses 700 MB more video memory than the game in Unity. At medium settings, the Unreal Engine game uses 800 MB more video memory than Unity. At maximum settings, Unreal Engine consumes 2,266 MB more video memory compared to Unity. For total video memory usage (figure 15), these differences amount to 875 MB, 939 MB, and 2,451 MB, respectively.

The chart presented in figure 16 shows that the game in Unreal Engine consumes 550 MB more RAM than the game in Unity at maximum graphics settings, 513 MB more RAM at medium graphics settings, and 552 MB more RAM at low graphics settings.

When considering the overall RAM usage (figure 17), it can be observed that at low graphics quality

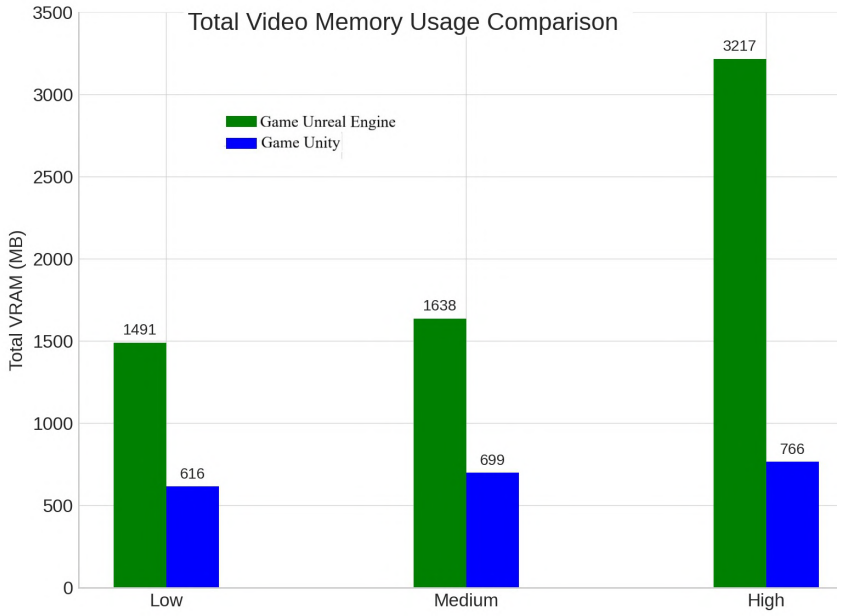


Figure 15: Total video memory usage at low, medium, and maximum game graphics settings.

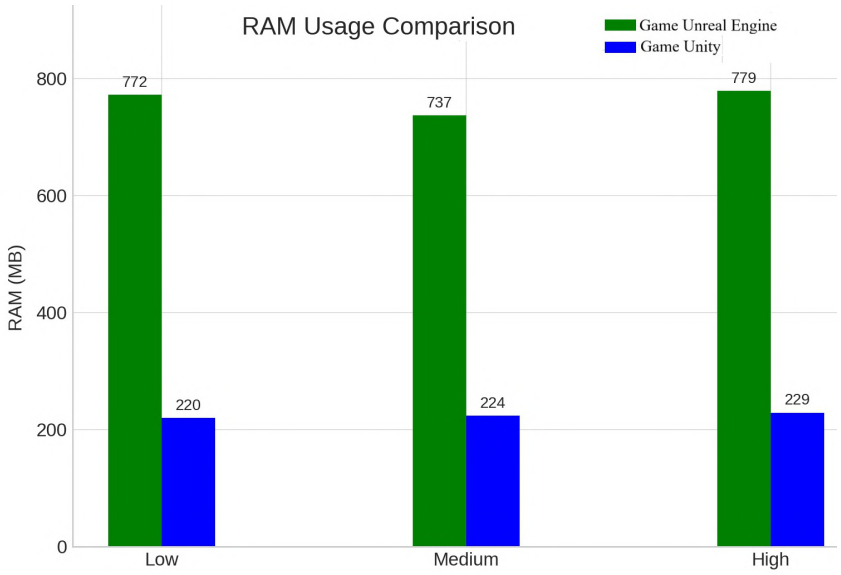


Figure 16: RAM usage by the game at low, medium, and maximum graphics settings.

settings, the difference is 832 MB, with the Unreal Engine game consuming more resources than the Unity game. At medium graphics quality settings, the difference is 524 MB, again in favor of Unity. At maximum graphics quality settings, the difference between the Unreal Engine and Unity games is 714 MB, once again in favor of Unity.

As can be observed, nearly all hardware resource parameters are higher when the game is run on the Unreal Engine, regardless of graphics quality settings – whether low, medium, or maximum.

5. Conclusion

This work implemented a computer game in the genre of a 3D platformer based on the Unreal Engine 5.4.4 and investigated the impact of games created on different game engines on computer hardware resources. Based on the conducted analysis, Unreal Engine 5.4.4 was selected as the most suitable

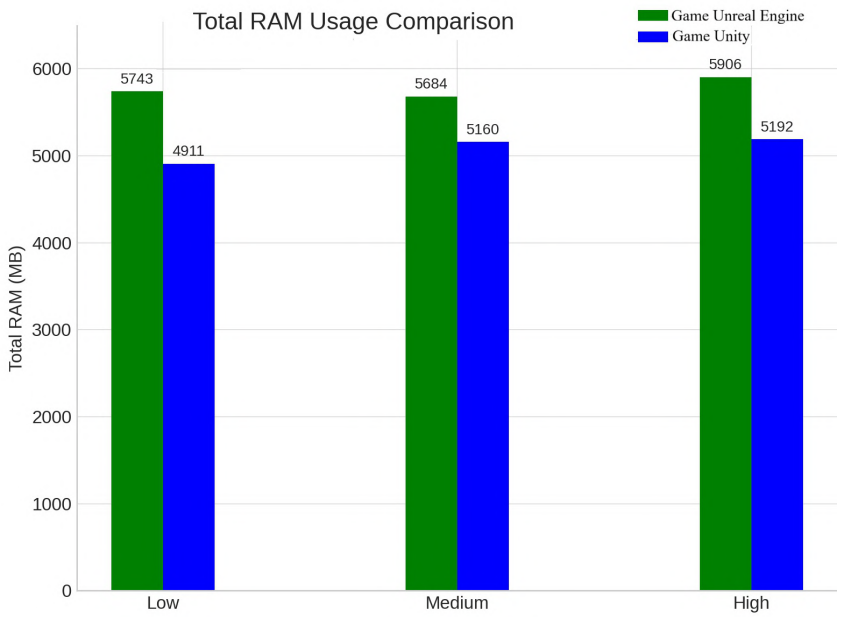


Figure 17: Overall RAM usage at low, medium, and maximum graphics settings.

platform for the implementation of a 3D platformer game, with Blueprint visual scripting chosen for developing game logic. A prototype game, The Ball, was developed to serve as a case study for assessing the performance differences between Unreal Engine and Unity 2021.3.15f1 under varying graphical settings. The game was designed with clear specifications, including defined genre, target audience, gameplay mechanics, objectives, and platform support.

A structured methodology was applied for creating game objects, levels, and user interfaces, utilizing Unreal Engine's tools such as Widget Blueprints, Level Blueprints, and Actor classes. Resource usage monitoring under different graphical configurations revealed that Unreal Engine generally demands higher hardware resources compared to Unity.

Specifically: Unreal Engine demonstrated significantly higher VRAM and RAM consumption across all settings; CPU load was consistently higher for Unreal Engine, albeit by a smaller margin; GPU load was also greater in Unreal Engine, with the difference narrowing under maximum graphics settings; Frame rates were noticeably lower in Unreal Engine, especially at lower and medium graphics settings, although both engines delivered playable performance within the acceptable range of 30–60 FPS. These findings indicate that while Unreal Engine delivers superior graphical quality and advanced capabilities for creating complex interactive environments, this comes at the cost of higher hardware demands. The trade-off between visual fidelity and resource efficiency is a critical consideration for developers when choosing an engine, particularly for projects with specific hardware limitations or performance targets. In summary, this research highlights the importance of a systematic approach to game engine selection, emphasizing the need to balance artistic vision, gameplay requirements, and hardware constraints. The comparative analysis conducted in this work provides valuable insights for both academic research and practical game development, offering a foundation for further studies on optimizing game performance across different platforms and engines.

Analysis of the collected data reveals that Unreal Engine 5.4.4 demonstrates higher graphical fidelity and smoother rendering performance at the cost of higher CPU and GPU usage. Unity 2021.3.15f1 exhibits lower resource demands but with a comparatively reduced FPS rate under equivalent conditions. Memory consumption differences were minimal but consistent across different levels and settings. These results highlight the trade-offs developers face when selecting a game engine. Unreal Engine provides superior visual quality and realistic physics but requires more robust hardware resources, whereas Unity offers a more lightweight solution suitable for lower-end systems or rapid prototyping. This aligns with prior studies indicating Unreal Engine's strength in high-fidelity rendering and Unity's advantage

in accessibility and performance efficiency. The findings of this experiment provide a foundation for making informed decisions about engine selection, particularly in projects where hardware constraints and graphical fidelity must be balanced.

Declaration on Generative AI

During the preparation of this work, the authors used ChatGPT in order to: grammar and spelling check. The use of AI tools was limited to supporting the writing process and did not influence the conclusions drawn from the evidence. After using this tool/service, the authors reviewed and edited the content as needed and took full responsibility for the publication's content.

References

[1] N. Faisal, M. Chadhar, A. Goriss-Hunter, A. Stranieri, Business Simulation Games in Higher Education: A Systematic Review of Empirical Research, Human Behavior and Emerging Technologies 2022 (2022) 1578791. doi:10.1155/2022/1578791.

[2] M. Tiahunova, O. Tronkina, G. Kirichek, S. Skrupsky, The Neural Network for Emotions Recognition under Special Conditions, in: S. Subbotin (Ed.), Proceedings of The Fourth International Workshop on Computer Modeling and Intelligent Systems (CMIS-2021), Zaporizhzhia, Ukraine, April 27, 2021, volume 2864 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2021, pp. 121–134. URL: https://ceur-ws.org/Vol-2864/paper11.pdf.

[3] G. Kirichek, S. Skrupsky, M. Tiahunova, A. Timenko, Implementation of web system optimization method, in: S. Subbotin (Ed.), Proceedings of The Third International Workshop on Computer Modeling and Intelligent Systems (CMIS-2020), Zaporizhzhia, Ukraine, April 27-May 1, 2020, volume 2608 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2020, pp. 199–210. URL: https://ceur-ws.org/Vol-2608/paper16.pdf.

[4] B. Fritz, Experiencing Gameworlds : Understanding Zelda and Mario Through the Lenses of Art History and Phenomenology, Ph.D. thesis, Lund University, 2025. URL: https://lup.lub.lu.se/search/files/215008893/Avhandling_Bj_rn_Fritz_LUCRIS.pdf.

[5] K. K. Liepa, Psyche as a Playable Construct in Video Games, Master's thesis, The University of Bergen, 2023. URL: https://hdl.handle.net/11250/3071930.

[6] Y. Zhu, D. Kim, A. Alsheimer, A Review of Game Design Techniques for Managing Suspense, in: A. L. Brooks (Ed.), ArtsIT, Interactivity and Game Creation, volume 479 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, Springer Nature Switzerland, Cham, 2023, pp. 174–186. doi:10.1007/978-3-031-28993-4_13.

[7] A. Konttinen, Core Features of 3D Platformers, Bachelor's thesis, University of Applied Sciences, Jyväskylä, 2024. URL: https://urn.fi/URN:NBN:fi:amk-2024122238039.

[8] A. Hadi, Designing Platformer Game using GameMaker Studio, International Journal of Art, Design, and Metaverse 1 (2023) 23–30. URL: https://topazart.info/e-journals/index.php/ijam/article/view/69.

[9] K. Marak, Independent horror games between 2010 and 2020: Selected characteristic features and discernible trends, Images. The International Journal of European Film, Performing Arts and Audiovisual Communication 29 (2021) 175–190. doi:10.14746/i.2021.38.11.

[10] A. W. Octaviani, I. Irfansyah, Formal Game Element Analysis of Rhythm Fighting Game, in: Proceedings of the ICON ARCCADE 2021: The 2nd International Conference on Art, Craft, Culture and Design (ICON-ARCCADE 2021), Atlantis Press, 2021, pp. 243–251. doi:10.2991/assehr.k.211228.032.

[11] N. Richard, The Art of Survival Design: A Blueprint to Survival Game Design Beyond the Genre, Bachelor's thesis, Tampere University of Applied Sciences, 2024. URL: https://urn.fi/URN:NBN:fi:amk-2024120231812.

[12] O. Poberailo, The genre specificity of interactive cinema, Academia Polonica 69 (2025) 64–74. doi:10.23856/6907.
[13] S. Rahimi, J. T. Walker, L. Lin-Lipsmeyer, J. Shin, Toward Defining and Assessing Creativity in Sandbox Games, Creativity Research Journal 36 (2024) 194–212. doi:10.1080/10400419.2022.2156477.
[14] R. P. De Lope, N. Medina-Medina, M. Urbieta, A. B. Lliteras, A. Mora García, A novel UML-based methodology for modeling adventure-based educational games, Entertainment Computing 38 (2021) 100429. doi:10.1016/j.entcom.2021.100429.
[15] D. Felini, Beyond Today's Video Game Rating Systems: A Critical Approach to PEGI and ESRB, and Proposed Improvements, Games and Culture 10 (2015) 106–122. doi:10.1177/1555412014560192.
[16] Unity Technologies, Unity Documentation, 2026. URL: https://docs.unity.com.
[17] Epic Games, Features – Unreal Engine, 2026. URL: https://www.unrealengine.com/features.
[18] J. Linietsky, A. Manzur, Godot community, Introduction – Godot Engine (stable) documentation in English, 2026. URL: https://docs.godotengine.org/en/stable/about/introduction.html.
[19] Redot community, Redot Docs – master branch – Redot Engine (latest) documentation in English, 2026. URL: https://docs.redotengine.org/.
[20] N. V. Moiseienko, M. V. Moiseienko, V. S. Kuznetsov, B. A. Rostalny, A. E. Kiv, Teaching computer game development with Unity engine: a case study, in: O. Y. Burov, S. H. Lytvynova, S. O. Semerikov, Y. V. Yechkalo (Eds.), Proceedings of the VII International Workshop on Professional Retraining and Life-Long Learning using ICT: Person-oriented Approach (3L-Person 2022), Virtual Event, Kryvyi Rih, Ukraine, October 25, 2022, volume 3482 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2022, pp. 237–251. URL: https://ceur-ws.org/Vol-3482/paper330.pdf.
[21] Y. Knirik, Build-the-ball-unreal-engine, GitHub, 2024. URL: https://github.com/Knirik-Yegor/Build-The-Ball-Unreal-Engine.
[22] Y. Knirik, Build-The-Ball, GitHub, 2023. URL: https://github.com/Knirik-Yegor/Build-The-Ball.
[23] Y. O. Knyryk, Development of a computer game in C# using the Unity environment, Bachelor's thesis, National University «Zaporizhzhia Polytechnic», Zaporizhzhia, 2023. URL: https://eir.zp.edu.ua/items/1d1d37be-9f76-402c-8531-4985f0978f23.