A Comparative Analysis of Energy Consumption Between Visual Scripting models and C++ in Unreal Engine: Raising Awareness on the importance of Green MDD

Javier Verón
jveron@usj.es
SVIT Research Group, Universidad San Jorge
Villanueva de Gállego, Spain

Carlos Pérez
cperez@usj.es
SVIT Research Group, Universidad San Jorge
Villanueva de Gállego, Spain

Coral Calero
Alarcos Research Group, University of Castilla-La Mancha
Ciudad Real, Spain

Mª Ángeles Moraga
Alarcos Research Group, University of Castilla-La Mancha
Ciudad Real, Spain

Francisca Pérez
SVIT Research Group, Universidad San Jorge
Villanueva de Gállego, Spain

Carlos Cetina
SVIT Research Group, Universidad San Jorge
Villanueva de Gállego, Spain

ABSTRACT

Video game engines are used in most modern video games because they simplify and speed up development. In addition, some of the most popular engines, such as Unreal Engine 5 (UE5), also integrate visual scripting tools. Visual scripting in UE5, through Blueprints, is a model-driven development approach that replaces text code, like C++, with a visual language of interconnected nodes representing functions and data flows, forming a flowchart-like logic diagram. This approach simplifies game development by abstracting complex code into intuitive, visual models, enabling creators to construct and iterate game components without extensive programming knowledge. Although Blueprint models usually decrease the complexity of implementing components, thus accelerating the development, they might lead to less energy-efficient runtime performance than C++. In this work, we evaluate the energy consumption of three relevant video game components (health system management, inputs processing, and collections operations for an inventory), each implemented with Blueprint models and C++. The results show that the energy consumption per frame when using C++ is up to 48% lower than when using Blueprint models. The combination of artistic and technical profiles in video game developments has favoured the adoption of Blueprint models. However, there is a lack of works analyzing the energy consumption. Until this work, there was no evidence that the success of models for developing video games, like the one under study in this work, was accompanied by a cost in energy consumption for certain situations. Given the huge popularity of video games, this cost in energy might reach up to the equivalent of the energy consumption of 28 million European households.

MODELS '24, September 22–27, 2024, Linz, Austria

ACM ISBN 979-8-4007-0504-5/24/09
https://doi.org/10.1145/3640310.3674099

CCS CONCEPTS

• **Software and its engineering → General programming languages**; **Visual languages**.

KEYWORDS

Energy consumption, Video Games, Green software, Green Video Games, Software sustainability, Game Engines, Unreal Engine, Software Models, Visual Scripting, Blueprints, C++, Game Software Engineering

ACM Reference Format:
Javier Verón, Carlos Pérez, Coral Calero, Mª Ángeles Moraga, Francisca Pérez, and Carlos Cetina. 2024. A Comparative Analysis of Energy Consumption Between Visual Scripting models and C++ in Unreal Engine: Raising Awareness on the importance of Green MDD. In *ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems (MODELS '24), September 22–27, 2024, Linz, Austria.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3640310.3674099

1 INTRODUCTION

Video game engines integrate many different assets for the development of video games, like a graphics engine or a physics engine, as well as many tools that wrap around them to accelerate development. The video game engine enables developers to create different games in a much more agile way, without having to repeat or code low-level elements whose design from scratch would be very costly and incompatible with the pace of development of these products. Although there are companies that develop their video game engines, it is more common to use an already existing engine from the market. Unreal Engine [19] is one of the most popular choices among developers.

One essential component of video game engines is software models. Developers have the option to create video game content either through direct coding (e.g., C++) or by utilizing the software models provided by the engines. While coding offers developers greater control over the content, software models provide a higher level of abstraction, distancing themselves from the underlying implementation and technology. This enables developers to work with concepts more closely related to the problem domain. Consequently, developers are freed from the intricate details of common low-level

implementations like those of physics and graphics, allowing them to focus on the content of the game. In 2014, Unreal Engine included its own domain-specific modeling language called Blueprints. Ten years after the launch, the adoption of Blueprints by developers is majority [9, 29, 30].

The gaming industry is steadily gaining prominence in the realm of computer-based activities. The evolution of recent video games is marked by, among other things, enhanced graphical realism and an increase in the number of interactions and players, needing more powerful equipment capable of managing this complexity without compromising user experience. This gain in computational complexity usually implies that the energy consumption and the environmental footprint of video games are also increasing.

An approximate estimation of the global video game energy consumption can be made by multiplying the number of worldwide video game players (3.38 billion according to [6]) times the average consumption per player. This later amount is obtained by multiplying the average hours a gamer plays per year (440 hours [8]) times an estimation of the power required by a representative video game, which we obtain from our measurements of the average power (358.6 W) required when playing *Baldur's Gate III* without PC background processes. To produce a more conservative estimation, we do not include console or mobile gamers in this estimation, only PC gamers are considered, which account for 43% of the total players [5]. As a conclusion, we have estimated the energy consumption of video games per year to be at least 230 TWh worldwide. This estimation assumes that all gamers played the same game. This figure is equivalent to the annual energy consumption of approximately 59 million European households (3900 kWh is the average annual amount of electricity purchased by an EU member residential electric-utility customer [1]). That means that improvements in video game software have enormous potential for energy savings. However, perhaps due to its relative youth, the video game sector has not yet developed the same level of environmental awareness as other computing technology sectors.

This work constitutes the first approach to evaluate the difference in energy consumption between using Blueprint models and using a more traditional coding approach in C++ for a video game. For enhancing result reliability, we employed the validated framework FEETINGS [24] for the measurement and analysis of the energy consumption, following the validated procedure included in the framework (GSMP) and employing the technological infrastructure that facilitates the capture and analysis of software energy consumption, which are accepted by the green computing research community [7, 14, 15].

We studied three components of video games in two different versions (using Blueprint models for one version and C++ code for the other one) to measure and evaluate their energy consumption. These three components are health system management, input processing, and collections operations for an inventory. Industry professionals have confirmed them as relevant in video games. Both versions of each component have also been developed by one company and then revised by other industry professionals.

The results show that C++ can have up to 48% less energy consumption than Blueprint models for the execution of some components. Software models (such as Blueprints) are proving to be a success in the gaming domain, but this success is accompanied by an unidentified energy consumption problem that this work reveals.

This work has the potential to encourage video game engine developers to improve the generation of code from models from a greener perspective. Furthermore, these results might also encourage a new branch of research on green model-driven video game development. Finally, this work raises awareness on the adoption of models being accompanied by green challenges, and this phenomenon is not only limited to the success of models in video games, but can also take place in more traditional software domains.

The paper is structured as follows: in Section 2 we provide a concise summary of video game engines, focusing on Unreal Engine 5 and how Blueprint models are integrated to foster video game development. In Section 3 we discuss how the energy consumption measurement approach is planned and performed following accepted practices by the green research community. In Section 4 we introduce the Research Questions and explain the implementation details. In Section 5 we analyze and discuss the results. The threats to the validity of the experiment are discussed in Section 6. In Section 7 we outline previous related work on the subject. Lastly, we present our summarized conclusions and future work in Section 8.

2 BACKGROUND

A game engine is a tool in the field of video game development, encapsulating a suite of integrated technologies and tools that streamline the production of games. It not only expedites the development process, but also enhances the quality and performance of the final product, thereby playing a crucial role in the evolution and proliferation of video games as a popular form of entertainment and artistic expression.

Game engines are specifically designed to enable and streamline the creation of video games, providing a layer of software components that are reusable across different video games, thus significantly reducing the time and resources required to develop a new game from scratch. These engines can be equipped with a wide array of functionalities including, but not limited to, rendering graphics (2D and 3D), simulating physics, managing memory and assets, and facilitating audio output, networking capabilities for multiplayer games, artificial intelligence for non-player character behavior, and tools for animation and environmental effects.

Additionally, many game engines offer more advanced features such as multi-platform building, which makes them an appealing choice if it is the case of a video game or project targeted for multiple platforms. With this feature, game engines allow game developers to focus on the design and gameplay aspects of the game rather than the intricacies of platform-specific development of these components.

Most game developers use game engines, (60% as of 2022 [11]) and there are multiple industry-scale engines. One of the most popular game engines among video game developers is Unreal Engine [11]. Video game developers have used Unreal Engine since its launch in 1998 until the current version of Unreal Engine 5 (UE5). Nowadays, UE5 is the preferred choice for high budget developments. In fact, nearly 80 of the most highly anticipated games released in 2023 are powered by Unreal Engine [16].

It is important to take into account that UE5 is not only one of the most used game engines among video games developers, but also millions of users play to video games powered by UE5: the video game *Fortnite* has more than 235 million users per month [2], and it is only one of the many games developed with this engine. Furthermore, the adoption of UE5 includes a wide range of applications beyond traditional gaming, including film and television, architecture, automotive, or manufacturing, and simulation [17], although their main focus is video game development.

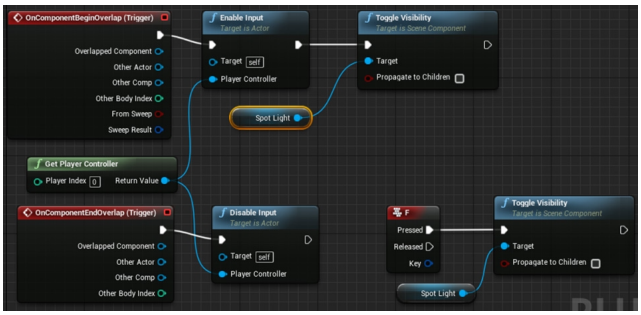

Figure 1: Blueprint models in UE5: Interactive Spotlight Control via Trigger

2.1 Unreal Engine 5 Blueprints

The software model of UE5 (a domain-specific modeling language) utilizes a node-based language called Blueprints, as depicted in Figure 1, where functions and actions are represented as interconnected nodes. It's crucial to distinguish between the concept of software model and that of a 3D model (or mesh). 3D models are commonly employed in the fields of video games and computer graphics to denote the visual representation of three-dimensional shapes.

Blueprints stands out as a feature that democratizes the game development process, allowing programmers and designers to create complex game logic and interactions without the need for traditional code-based programming.

In the development of video games, a seamless fusion of artistry and programming expertise is essential, resulting in a product that reflects the collaborative endeavor of development teams conformed by diverse profiles. According to a recent industry survey [33], these teams are formed by software developers (24%), game designers (23%), artists (15%), UI designers (8%), and QA engineers (5%). The diversity within these teams fosters a predilection for utilizing software models. Software models offer a higher level of abstraction than coding, thus facilitating greater involvement from members less focused on technical aspects. This strategy not only democratizes the development process but also enriches the creative input across various roles.

The Blueprint in Figure 1 is designed to control input and visibility of a spot light, based on the player's overlapping with a trigger area. When the character controlled by the player enters the trigger area, input is enabled, allowing them to interact with the spot light by pressing the the specified key 'F'. When the key is pressed, the visibility of the spot light is toggled, meaning it will turn on or off. Exiting the trigger area disables the input, preventing the player from toggling the visibility of the spot light until they re-enter the trigger area.

This logic implementation approach significantly lowers the entry barrier for game development, enabling individuals with limited coding knowledge to contribute effectively to the development process instead of needing to learn how to program in C++, which is the alternative and more traditional option for logic implementation in UE5. Moreover, Blueprints facilitate rapid prototyping and iteration, as changes can be made and tested in real time, thus accelerating the development cycle and encouraging experimental gameplay designs.

Blueprints need to be compiled into UnrealScript VM bytecode which will run on a virtual machine when used in-game [18]. The difference in the logic of the game between C++ code and Blueprint models might not be noticeable in terms of behavioural results; however, the impact difference in terms of energy consumption between using C++ or Blueprint models has not been explored so far.

3 STUDYING THE ENERGY CONSUMPTION OF SOFTWARE

If we want to rigorously study the environmental impact of software, it is essential to know its energy consumption while in operation as precisely as possible.

Two kinds of approaches serve this purpose: The first one uses specialized software tools capable of estimating hardware energy consumption during software execution, such as PowerAPI [20] or Joulemeter [23]. These tools do not directly measure energy consumption but instead derive estimates from a model fed with parameters collected by the software during execution. Despite their ease of use and ability to yield valuable results across various levels of detail, their reliance on estimation implies that result accuracy eventually depends on the model approximations.

Alternatively, the other approaches involve employing measuring equipment linked to a suitably instrumented computer to directly record energy consumption during software execution. This approach offers the advantage of delivering genuine energy consumption data, whose quality only depends on equipment quality and measurement protocol. Nonetheless, it requires computer instrumentation for testing and may prove to be more expensive and difficult to implement.

Merely possessing a physical device to measure computer energy consumption during software operation is not adequate to ensure conclusive results. A comprehensive process is imperative to ensure study rigor and consistency, thereby rendering obtained conclusions to be scientifically relevant.

In this paper, we have employed the Framework for Energy Efficiency Testing to Improve Environmental Goals of the Software (FEETINGS) [24]. This framework is accepted by the green computing community [7, 14, 15] and offers the following assets:

- A standardized terminology for measuring software energy consumption.
- A green software measurement process (GSMP), accessible as an electronic guide (https://alarcos.esi.uclm.es/FEETINGS/),

designed to systematically support the necessary tasks for measuring software energy consumption.
- A technological infrastructure that facilitates the capture and analysis of software energy consumption. This infrastructure includes an Energy Efficiency Tester (EET), serving as a measuring instrument, and a software tool (ELLIOT) designed for visualizing and analyze the energy consumption results captured by the EET automatically. The EET, a hardware-based device, has been utilized instead of software estimators to provide more realistic and accurate energy consumption values. It captures consumption data directly from the power supply of the PC, offering a sampling frequency of 100 Hz. The EET measures the consumption of the entire PC, referred to as the Device Under Test (DUT), as well as various components of the PC, including the processor, graphics card, hard disk, and monitor.

Monitor	Falkon Q2702S 27" 2K
Motherboard	Asus Prime B460-Plus
Processor	i7 10700
RAM	4 modules of 32GB DDR4 Kingston 2666MHz CL16
Graphics card	Zotac Gaming GForce RTX 3060 12 GB GDDR6
Hard Disk	Hard Disk & Kingston SSD A400 – 480GB SATA
Power supply	Energy PS901SX 900W
O.S.	Windows 11 Pro

Table 1: DUT specifications

The GSMP process comprises seven phases, covering all the essential steps for conducting a comprehensive analysis of software energy consumption during software execution:

(1) **Scope definition**. The main goal of this phase is to obtain a complete specification of the requirements for the evaluation of energy efficiency, including the precise definition of the software to be analyzed and the development of the different test cases to be run. The test cases for our work are explained in detail in Section 4.1.
Our study aims to compare the energy consumption of three components representative of real video games, implemented with two different approaches (Blueprint models and C++ code). It is worth emphasizing that our objective is to find out if there is any significant difference in energy consumption, not to discover what this difference is exactly, as this will always depend on the environment in which the applications are run.

(2) **Measurement environment setting**. This phase includes activities such as the selection of the measurement equipment to be used, with the detail of its technical specifications; the decision of the physical magnitudes to be measured, and the verification that there are no background processes that may contaminate the desired results.
The measurement environment used in this work is the offered one in the FEETINGS framework, which includes: the EET device, the ELLIOT software, and the DUT. The DUT is a desktop computer utilized for executing the test cases and conducting time and power measurements. In our experiment, the DUT had the specifications outlined in Table 1.
In green software research, accepted metrics include the energy use of the processor, the hard disk drive (HDD), and the total energy consumption of the device [7]. However, assessing the energy use of the graphics card becomes essential for video games, given its intensive utilization in gaming.
To take care of the possible contamination of background processes necessarily running in the operative system, the final power measurements are acquired by subtracting the baseline power required (measured before launching each application under study) from the measurements taken during the experiment (measured while executing each application).

(3) **Measurement environment preparation**. This stage includes the elimination of services or processes running in the background of the DUT, the setting of the sample size (since measurements must be repeated to guarantee the representativeness of the results), and the configuration of the testbed.
The EET replaces the original power source of the DUT where the software runs with its own power supply. This way, the power required and the execution time are measured and recorded. To guarantee the reliability of the analysis and the statistical results obtained, all the test cases executions were recorded 30 times. Also, in our study, the duration of the execution and recording was 30 seconds in all of the test cases at a sampling frequency of 100 Hz and the execution of every test case did not require any actions performed by a human, as the actions of each test case are performed automatically.

(4) **Performing the measurements**. Measurements are performed in the sequence defined in the previous phase, and all the raw data are collected and recorded.

(5) **Test case data analysis**. It covers the analysis of the previous data, checking for incorrect values and eliminating them from the dataset, and performing the statistical analysis of the data.
A measurement is considered invalid if it is recorded from a wrong execution, identified by its inconsistency with the results from the other executions; or if it qualifies as an outlier, where one or more values are significantly higher or lower than those from other executions. This processing is computed by ELLIOT, which also calculates the statistics of the values obtained.

(6) **Software entity data analysis**. It includes the calculation of the energy consumed by the execution of the software by subtracting the appropriate baseline consumption, and the analysis and interpretation of the results, to make sure they address the research questions that were put forward at the beginning of the research.
In this work, the analysis and interpretation of the data are presented in Section 4.2 and Section 5.

(7) **Reporting the results**. Finally, phase 7 deals with appropriate reporting of the exercise, to ensure that the gained knowledge is available and both the measurement process and results can be replicated by other researchers.

4 EVALUATION

This section includes the research questions that we aim to answer, the implementation details, and the results obtained.

This study aims to examine the impact on the energy consumption of two different approaches for implementing logic in a video game (i.e. Blueprint models and C++), but video games integrate many complex components to create immersive and interactive experiences. If we evaluated the consumption of a complete video game, we would not be able to isolate specific components, as video game scenes often combine many of these at once. These components encompass graphics rendering, physics engines, audio processing, health system management, input processing, and inventory[1] management, among many others.

Some of these components, like graphics rendering, physics, and audio processing, are already managed by game engines themselves, such as UE5. However, some other components need to be implemented by the game developers. This is the case of health system management, input processing, and inventory operations. These components are also three of the most common components in video games, which makes them representative cases for the implementation comparison in this work.

Figure 2: Inventory system in the video game *The Witcher® 3: Wild Hunt*

An example of the usage of an inventory system is shown in Figure 2. The inventory screen in this Figure displays items which the players can use on their character. On the left side, materials for different interactions like crafting and making and modifying equipment are displayed. The center shows the weapons of the character, like swords and a crossbow, along with some number consumable objects and other fighting elements like bombs. The right side showcases the current armor of the character and other equipment, as well as the health status. Between the center and the right side, there is a representation of the character with the currently equipped weapons and armor. The inventory helps players to manage resources and gear within the game.

An example of the usage of a health system and input processing is shown in Figure 3. This Figure shows the main character, controlled by the player, in a natural setting. The red bar at the top left is the health bar, indicating the life energy of the character. When this bar empties, the character can no longer continue and the game is restarted from a previously saved point. The player moves and orders the character through this virtual world using various inputs from a keyboard or a controller, as indicated by the buttons on the bottom right of the screen. These would correspond to buttons on a game controller for actions like moving, attacking, or dodging. Both Figure 2 and Figure 3 are taken from the industrial case *The Witcher® 3: Wild Hunt*, a blockbuster video game which had sold more than 50 million units by the first quarter of 2023 [13].

We aim to compare the energy consumption of video games with a different logic implementation approach: one using Blueprint models and the other one using C++ programming. In order to tackle this, we formulated the following Research Questions:

- RQ1 Does implementing a health system management component for a video game with Blueprint models entail more energy consumption at run time than with C++?
- RQ2 Does implementing an input processing component for a video game with Blueprint models entail more energy consumption at run time than with C++?
- RQ3 Does implementing an inventory search component for a video game with Blueprint models entail more energy consumption at run time than with C++?

Answering each of the Research Questions will lead us to compare the energy consumption difference between the two different logic implementations of each of the three components, allowing us to conclude if there is a difference in energy consumption between using Blueprint models and using C++ programming or not.

4.1 Implementation Details

For this study, we need six test cases that allow us to compare the measurements taken from both logic implementation approaches under study, i.e., we had to implement three components in two different versions each: one version using Blueprint models and the other version using C++ code. The components implemented are the following: health system management, input processing,

Figure 3: Health bar at the top left of the image and main character moving with the input of the player in the centre of the image in the video game *The Witcher® 3: Wild Hunt*

[1]Inventories are collections of game elements that the player can manage. This results in the need of implementing different operations, including search operations and memory management.

and inventory operations. The outcome of implementing each component in both versions is a total of six test cases (three of them implemented with Blueprint models and the other three in C++).

Currently, there are no publicly available video games with two versions which are identical in every way except for the implementation approach used (Blueprint models or C++ code). Therefore, the implementations were performed in collaboration with a professional video game development company. The components were implemented by the company using the latest version (5.3.2) of Unreal Engine, which in turn used the version 17.6.5 of Microsoft Visual Studio Community 2022. Each version of each component was implemented by the professional developers in the company and then reviewed by other professional developers unaffiliated to the company, who agreed on the way to develop it.

The six applications were compiled in release mode with default shipping settings for Windows, since it is the way that would reach end users and a video game is expected to be run a much higher number of times by all end users than by developers. Also, the six applications built in UE5 are not running only the component under study. The applications require a minimum scenario for the execution of the components to be as similar as possible to how these components would appear in an industrial case. The minimum scenarios have other components working; like a physics system, or graphics rendering; in order to conform a complete application built with UE5. All of these components, except for the one under study, are similar between the applications containing each of the two versions of the components under study.

Figure 4 shows a screenshot of each one of the three different scenarios, each containing one of the components under study. Some of the additional components can be seen in this Figure, such as a character (grey humanoid) and an environment (sky and floor).

For the implementation of the health system management component, displayed in the left image of Figure 4, the character enters a damaging zone and its health gets decreased in time. In the Blueprint models version, the variables for the maximum and current health points are assigned and set in a Blueprint class named *AC_HealManager*. Then, another Blueprint class named *DamageZone* reduces the current health in *AC_HealManager* over time when the character is situated in the set damaging area. In the C++ version, the same logic design is followed in the C++ classes: *UHealManager* sets the maximum and current health and *ADamageZone* reduces the current health stored in *UHealManager* over time when the character is situated in the set damaging area. Snippets of both implementations approaches are shown in Figure 5 for this component.

For the implementation of the input processing component, displayed in the center image of Figure 4, the character moves when an input associated with the movement is pressed in a keyboard or a controller. The reading of the values of the inputs is implemented in both versions with the input system *Enhanced Input* of UE5, as usually done in commercial video games. In the Blueprints version, the association between the inputs values and the movement of the character is controlled in a Blueprints class named *BP_CustomPlayer*. In the C++ version, the movement of the player is controlled in the C++ class *AcustomPlayer*, which binds the input actions with the functions that make the player move. Snippets of both implementations approaches are shown in Figure 6 for this component.

For the implementation of the inventory operations component, displayed in the right image of Figure 4, both versions first load an inventory with 157 items. Then, an item is searched in that inventory periodically. In the Blueprints version, every item in the inventory is an instance of a Blueprints class named *BP_ItemData* and the search in the inventory is performed by using the UE5 Blueprint node *Find*, which searches for a specific item in a Blueprint Map with a provided key and returns the item if it is found. In the C++ version, the items are instances of a struct named *FDataItem* and the search in the inventory is performed in the C++ class *UInventoryManager* by calling the function *Find* of the Unreal Engine 5 class *TMap*, which returns a pointer to the value of the element if the map contains the key. Snippets of both implementations approaches are shown in Figure 7 for this component.

All Blueprints have been compiled with the default compiler

All six applications; i.e., the applications containing the two versions of each of the three components, implemented with Blueprint models and C++ code; are available at the following URL for replication purposes, along with the source code of the Unreal Engine 5.3.2 projects: https://zenodo.org/records/12570903

4.2 Results

TestCase	HDD	GraphicsCard	Processor	DUT
Health system (Blueprints)	1.73	100.68	31.14	324.38
Health system (C++)	1.73	100.85	30.86	330.62
Power diff	0.03%	0.17%	-0.93%	1.89%
Input processing (Blueprints)	1.66	101.59	45.88	370.87
Input processing (C++)	1.75	102.32	45.75	376.07
Power diff	4.81%	0.71%	-0.27%	1.38%
Inventory ops. (Blueprints)	1.77	101.30	45.33	375.04
Inventory ops. (C++)	1.73	100.70	45.44	374.53
Power diff	-2.19%	-0.60%	0.24%	-0.14%

Table 2: Power required (W) for the execution of the applications for 30 seconds and power difference comparison (C++ - Blueprints)

Table 2 shows the power required for executing the six applications for 30 seconds each. These results show that differences are minimal between the usage of Blueprint models and C++ code for all three components. The power required by the hard disk is between one and two orders of magnitude lower than that of the graphics card and processor, and the difference between both implementation approaches in the graphics card and the processor is under 1%. These results entail that, if we measure the power used over time (i.e. W.s), the answers to RQ1, RQ2, and RQ3 are similar: Implementing in Unreal Engine 5 any of the three tasks under test using Blueprint models results in similar energy consumption at runtime than implementing them with C++.

When measuring traditional software, only its energy consumption on W.s is measured [7]. However, we argue that to have a better understanding of energy consumption in video games, frame rate needs to be taken in consideration. As video games are real-time applications which require continuous visual feedback for the player, refresh rate in this type of applications is key. This refresh rate is

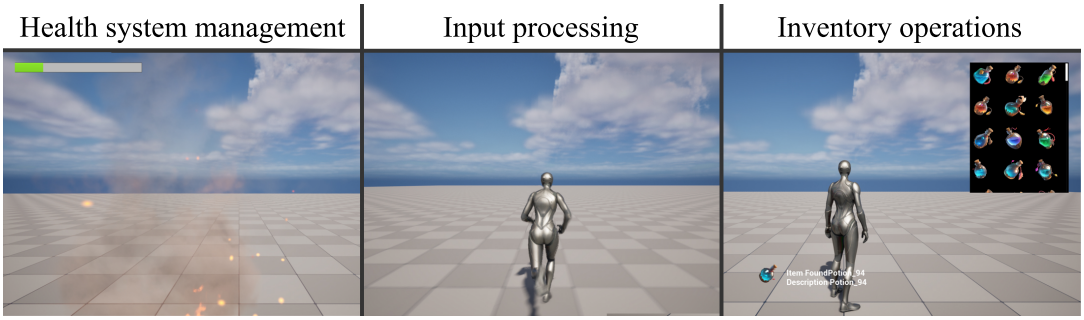

Figure 4: The three components implemented in UE5: health system management, input processing, and inventory operations.

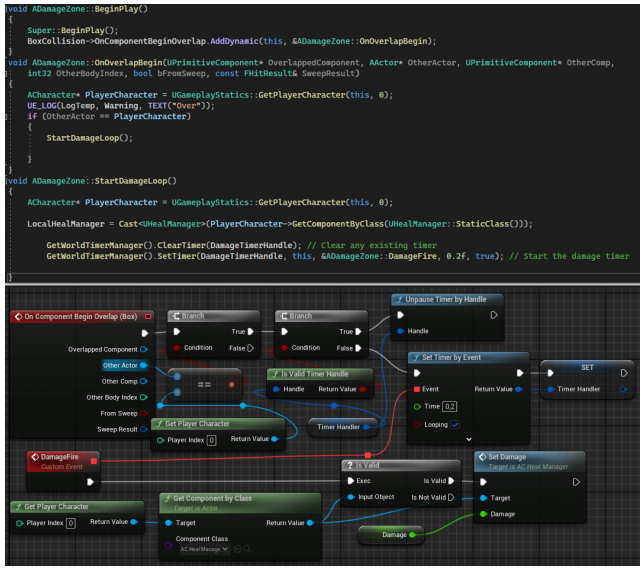

Figure 5: C++ and Blueprint models snippet where the character receives a decrease in current health when damaged.

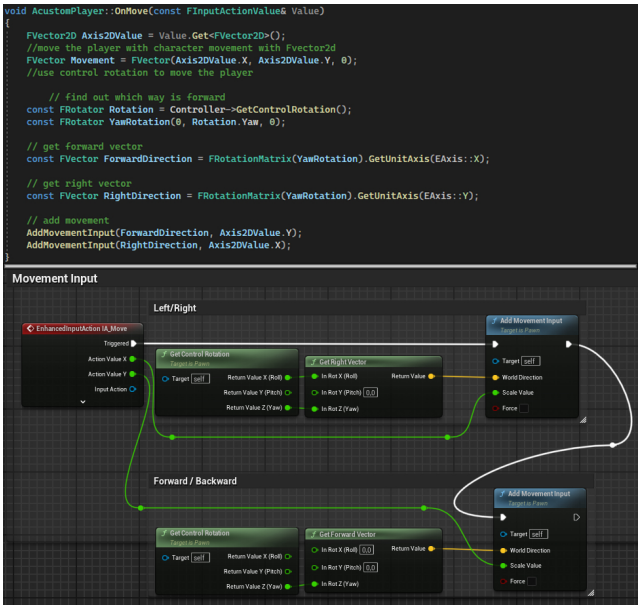

Figure 6: C++ and Blueprint models snippet where the input is processed for moving the character.

usually reflected in the frame rate or frames per second (FPS) in this context. FPS represent the amount of frames (i.e. images) that the application is able to render per second, with the consequent logic for updating the game world that is being rendered. The more FPS, the more fluid the gameplay will be, improving the user experience and the quality of the gameplay. Frame rate is so important that platforms owners like Sony or Nintendo will only allow developers to publish their games on their platforms (e.g., PlayStation 5 or Switch) if they achieve a minimum frame rate at any given time of the gameplay.

To take into account the FPS, we additionally measured the time in milliseconds that frames required in average during the executions of the applications described in Section 4.1. The results of this additional measurement, which are displayed in Table 3, show that the C++ implementation of the Health system component takes a 5.81% less time to render a frame, which means that the Blueprint models implementation of this component can achieve an average of 5.81% less FPS. The C++ implementation of the Input processing component, in contrast, takes a 1.73% more time to render a frame, which means that the Blueprints models implementation of this component can achieve an average of 1.73% more FPS. Finally, the inventory operations component is the component where the biggest difference between both implementation approaches is found: The C++ implementation of the inventory operations component takes 47.73% less time to render a frame, which means that the Blueprints models implementation of this component can achieve an average of 47.73% less FPS.

Test Case	C++ (ms)	Blueprints (ms)	Time diff.
Health system	0.78	0.82	-5.81%
Input processing	1.07	1.05	1.73%
Inventory ops.	0.66	0.98	-47.73%

Table 3: Average time required (ms) for the rendering of a frame of the applications and time difference comparison (C++ - Blueprints)

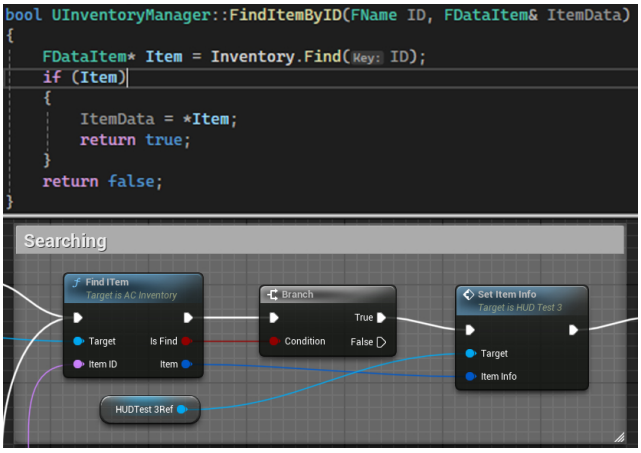

Figure 7: C++ and Blueprint models snippet where an item is searched in the inventory.

We showed in Table 2 the measurement of the power consumption of each implementation approach of each component in 30 seconds of execution, and in Table 3 the average time required for rendering a frame. With these measurements, we can compute the power consumption values per average frame. These values are displayed in mW for each version of each component are displayed in Table 4.

TestCase	HDD	GraphicsCard	Processor	DUT
Health system (Blueprints)	0.04	2.53	0.78	8.16
Health system (C++)	0.04	2.39	0.73	7.84
Power diff	-6.03%	-5.88%	-7.05%	-4.06%
Input processing (Blueprints)	0.05	3.28	1.48	11.96
Input processing (C++)	0.06	3.35	1.50	12.31
Power diff	6.22%	2.19%	1.21%	2.84%
Inventory ops. (Blueprints)	0.05	3.02	1.35	11.19
Inventory ops. (C++)	0.04	2.03	0.92	7.56
Power diff	-50.96%	-48.62%	-47.38%	-47.94%

Table 4: Power required (mW) per frame on average for the applications and power difference comparison (C++ - Blueprints)

Therefore, the actual answers to the Research Questions are the following:

Answering RQ1 Yes, implementing a health system management component for a video game with Blueprint models entails more energy consumption at run time than with C++. Specifically, implementing this component with C++ requires 5.88% less graphics card consumption and 7.05% less processor consumption than with Blueprint models.

Answering RQ2 No, implementing an input processing component for a video game with Blueprint models does not entail more energy consumption at run time than with C++. Specifically, implementing this component with C++ requires 2.19% more graphics card consumption and 1.21% more processor consumption than with Blueprint models.

Answering RQ3 Yes, implementing an inventory search component for a video game with Blueprint models entails noticeably more energy consumption at run time than with C++. Specifically, implementing this component with C++ requires 48.62% less graphics card consumption and 47.38% less processor consumption than with Blueprint models.

5 DISCUSSION

To understand why there are differences in energy consumption, we analyzed the open information about the UE5 implementation. UE5 has a lot of documentation about the blueprint model classes and about the C++ classes. All the C++ and Blueprints classes used to implement the components in this paper have their documentation online. However, the analysis of what is publicly available does not provide any explanation to justify the energy differences.

Our intuition is that the differences in power consumption are caused during the compilation of the Blueprint model to code. This compilation is outlined in the Unreal Engine 5 documentation [18]. In other words, the documentation describes the main steps of the compilation but does not provide the actual transformation between Blueprint models and code. We expect that, as works like this one raise awareness about the energy consumption of video games, video game engine developers start paying attention to which parts of the model transformation are influencing the energy consumption. Possibly, opening the model transformation will also help the game engine developers community and the research community to advance on improving the energy consumption.

It is also necessary that the documentation informs developers about the energy consumption implications carried by using different Blueprint concepts (nodes such as Find Element in the Array class) for the implementation. For that purpose, it is essential to keep researching to understand the differences in the consumption of Blueprint classes and the interactions among them. Currently, there is nothing similar in the official documentation because of the current lack of awareness about the energy consumption of video games.

Currently, Unity is the other most used video game engine currently in the video game industry [11]. Unity stands out because it offers a versatile performance range tailored to various platforms, making the development of mobile and XR games more appealing compared to Unreal Engine. Unity relies on C# as programming language for video game developers, but Unity seems to be following Unreal Engine steps in the adoption of Model-Driven Development (MDD). In 2017, the company Ludiq released Bolt: a domain-specific modeling language for Unity via the Unity Store. In 2020, Unity Technologies (the developers of Unity) acquired Bolt and, a year later, in 2021, Unity released its official Unity Visual Scripting language based on Bolt. Nowadays, Unity Visual Scripting is an integral part of Unity.

Our work reveals that the success of Blueprint models for developing video games with Unreal Engine comes with a penalty in energy consumption. This suggests that Unity Visual Scripting might also require attention. We have consulted the Unity Visual Scripting documentation and found no public information on the transformation of Unity Visual Scripting to code, similar to the lack of public information on the transformation of Blueprint models.

In fact, the transformations of video game software models to code should represent an opportunity to achieve the green perspective. Model-to-code transformations can embed green best practices to ensure their utilization by construction rather than solely depending on the developer's skill. Advances in green practices could be embedded into new versions of transformations, facilitating adoption by the developer community. To achieve this, it is necessary to raise awareness of the energy consumption of video games.

With the present work, we intend not only to raise awareness of the lack of research on energy consumption in MDD for video games but also to encourage and ease other researchers to perform experiments on other engines, like Unity. We release our implementations not only for replication purposes but also for extensibility. We expect other researchers to leverage the design of the components in this work and the rationale on the importance of FPS to better understand energy consumption in video games.

6 THREATS TO VALIDITY

We use the classification of threats to validity of Wohlin et al. [32].

Construct validity: This aspect of validity reflects "the degree to which the independent and the dependent variables are accurately measured by the measurement instruments used in the experiment".

Dependent variables: The power required for executing the different test cases is taken as the dependent variable. It was measured using the Efficient Energy Tester, validated as a dependable device for assessing the energy efficiency of software execution [7]. Additionally, each execution was repeated 30 times, and all applications containing the components under study were executed on the same physical machine, operating system, and configuration.

Independent variables: The independent variables are the use of Blueprint models or C++ to implement the different behaviors within Unreal Engine 5 and the three behaviors selected as representative of a typical game which consumption is to be assessed.

Internal Validity: As we try to determine if there is a causal relation between implementation approach (Blueprint models or C++) and energy consumption, it is important to check if the factor being investigated might be influenced by unaccounted variables.

Due to the nature of the experiments, all of the variables were controlled, thereby minimizing potential threats to internal validity. Specifically, the test cases are the same and run the same components in both approaches. It is worth noting that the repetition of the experiments 30 times during the same time also helped to minimize any possible bias. In addition, subtracting the baseline power in the measurements and removing invalid measurements (wrong executions or outliers) also helped to ensure that the dataset was valid for analyzing and answering the Research Questions.

External Validity: This aspect of validity focuses on the extent to which findings can be generalized and are relevant to other cases. Enhanced external validity indicates a greater ability to apply the results of an empirical study to real-world software engineering practices.

To minimize the risk of the Blueprints and C++ implementations not being comparable, the components have been implemented in collaboration by a professional video game development company specialized in development with Unreal Engine who are equally experienced with both C++ and Blueprints. Furthermore, the representativeness of the components as relevant components of video games that are present in the vast majority of video games being marketed today has been validated by this same company and two more additional professionals in the industry.

Another threat to the external validity is the fact that the measurements are obtained for a specific computer (the DUT being used). We mitigated this threat by employing a pre-validated procedure and measuring tool, enhancing result reliability. Even if consumption data were to vary across devices, our results indicate that the percentage variances between those values would be similar to those obtained in this case.

In addition to the classification of Wohlin et al. [32], it is important to remember that many components run simultaneously for a video game to function correctly. Although the three components under study were selected by a professional video game development company as frequently used, they may not be the most relevant in terms of power usage. Further studies with industrial case studies will be required to shed more light on this in the future.

7 RELATED WORK

Projections suggest that by 2025, around 20% of global energy usage will stem from information and communication technologies [3]. Consequently, the environmental footprint of this sector is significant and poised for further expansion in the foreseeable future. One of the sectors within these technologies where growth has been most evident in recent years is video games.

However, perhaps due to its relative youth, the video game sector has not yet developed the same level of environmental awareness as other computing technology sectors. In the case of computing technologies in general, there are research efforts in the area of green cloud computing [4], green mobile computing [28], green software [25], and green data centers [31] among others.

The research discussed in the work of Gutierrez et al. [15] emphasizes two approaches to examining energy usage: Green-IN and Green-BY. The former concentrates on enhancing energy efficiency within a specific domain (such as video games), while the latter focuses on utilizing a domain (like video games) to promote sustainability across various contexts.

Within the subcategory of "green-BY", it is where the research efforts are concentrated. For example, Johnson et al. [21] conducted a systematic review to assess the effectiveness of gamification and serious games in impacting domestic energy consumption.

Within the subcategory of "green-IN" computing, the focus has mainly been on studies analyzing the impact of hardware improvements on the environment. In recent years, the impact of proper software use (green-IN software) on the environment has also been recognized, and more effort is being devoted to its study [15].

However, there are hardly any studies that analyze the energy consumption of video game software despite its importance. A survey on "green video games", which is defined as the field of Green Computing that deals with the particularities of video games, is presented in the work of Pérez et al. [26]. Three main issues are identified in the survey by Pérez et al. [26] related to Green Video Games:

- *Battery life:* The surge of mobile devices as gaming platforms, coupled with the heightened complexity and graphic quality of applications, has underscored the constraints imposed by battery life on the gaming experience. Consequently, several studies are dedicated to enhancing the energy efficiency of video games, driven less by environmental considerations and more by the recognition of how reduced energy consumption prolongs battery life and, consequently, enhances the user experience.
- *Balance between energy consumption and various factors like application performance, user experience, and economic costs:* Several methods aimed at lowering energy usage in computer applications, notably in video games, such as Dynamic Voltage and Frequency Scaling (DVFS), may inadvertently compromise performance or user satisfaction, while others might affect economic considerations. Consequently, numerous studies concentrate on assessing the optimal parameter combination to achieve maximum energy savings with minimal deterioration in application performance, user experience, and acceptable costs.
- *Modelling and measurement of energy consumption:* Enhancing software energy efficiency requires a thorough understanding of the impact of each element on the overall energy consumption. Research endeavors aimed at quantifying software energy consumption under controlled circumstances and modeling its behavior in the pursuit of more energy-efficient versions of applications like video games.

The survey shows that there are studies on the energy consumption of video game software focusing on algorithmic efficiency. However, the survey does not identify any work that studies the influence of software models (such as Blueprints) on energy consumption as this work does.

The energy consumption of two of the most popular video game engines (Unity and Unreal Engine) is studied in the work of Pérez et al. [27], comparing their performance in three different scenarios representative of typical video games. The authors conclude that, since each of the engines outperforms the other in energy efficiency depending on the scenario, there is interest in finding the most energy-efficient video game engine technology.

Although the comparison focuses on the influence of game engines, it does not consider the influence of video game engine software models (Blueprints in Unreal Engine) on energy consumption as this work does. Considering the influence of video game software models on energy consumption is important because (1) they are increasingly used in video game development, and (2) they have higher energy consumption than their coding alternative, as this work shows.

In the broad sense of sustainability, there is a first wave of work that discusses the role that software models can play to achieve sustainability. The Sustainability Evaluation ExperienceR (SEER) framework [22] argues that software models are one of the main enablers to achieve sustainability in the context of self-adaptive systems. Gramelsberger et al. [12] propose a DSL model to describe sustainability indicators (eg., consumption types, CO2 emissions, or landscape usage). This DSL can be combined with ADLs to facilitate informed sustainability decision-making throughout the lifecycle of systems during their development. In the context of digital twins, David and Bork [10] provide an initial taxonomy that focuses on the technical dimension of sustainability. All of these works make the case that the modeling community is relevant to tackle the challenge of sustainability. Our work goes beyond in the sense of providing the first quantitative evidence of the importance of green MDD. It is true that software models can help to achieve sustainability, but it is also true that the very software models can pose a challenge to sustainability if energy consumption goes unnoticed.

An informal survey of the papers published in the MODELS Conference Proceedings and in the SoSyM Journal with the terms *consumption OR energy OR green* shows that the topic of energy consumption has not received sufficient attention from the community. As with video game models, models of other classical domains may also influence energy consumption. This is especially relevant in the era where low-code and no-code approaches are proliferating in industry as is the case of Microsoft Power Apps, Amazon Honeycode (recently ended or hiatus), Outsystems, or Google AppSheet.

8 CONCLUSION

The video games industry has experienced a significant expansion in user numbers and game sophistication in recent years. However, the environmental awareness in the video games industry, particularly the efforts to reduce the energy consumption, is still an under-explored field of research that requires more work on it, as the magnitude of the energy consumed in this sector makes this a relevant issue for the research community.

Unreal Engine 5 is one of the most used game engines in the industry and powers some of the most played video games in the present, which highlights its significance in this field. This engine offers two different logic implementation approaches to developers: Blueprint models and C++.

This work explores how different these implementation approaches are in terms of energy consumption. We selected three representative components to measure the differences in energy use between both implementation approaches: health system management, inputs processing, and inventory operations. The power required for the different implementation approaches has been measured in a controlled environment for all of the components. We employed the framework FEETINGS, which is accepted in the green computing community. This framework provided us the systematic process which we followed for the measurement and the technological infrastructure for capturing and analyzing the energy consumption.

When analyzing the power required per frame, we find that in two out of the three components (health system management and inventory operations) Blueprint models consume more energy per frame than C++ code. The inventory operations component is particularly interesting, showing nearly 50% less energy consumption per frame with the C++ implementation compared to Blueprint models. In the one component where C++ code consumes more power per frame (inputs processing), the difference is between 1% and 3% for both the graphics card and the processor.

Furthermore, there are still relevant common video game components to analyze in future works, such as networking, save and load system, dialogue system, particle system, camera behaviour, and AI path-finding navigation. Also, video game engines enable

deployment of video games across multiple target platforms (e.g., PC, mobile, VR, consoles), as it widens the audience reach for developers. This cross-platform development may reveal new insights into how models influence energy consumption in different devices, thus making the analysis of the same game components in different platforms another goal for our future work.

This work reveals that the success of software models in the video game domain comes with an energy cost. Due to the popularity of video games, this energy cost can be very high, equivalent to the consumption of millions of households. Another leading engine in the video game industry, Unity, is following the MDD (Model-Driven Development) steps of Unreal Engine, reinforcing the importance of considering the green perspective in MDD for video games. Moreover, the use of MDD outside the video game domain could impact energy costs in other areas. This work could prompt the video game industry and researchers to consider the green perspective and trigger a new branch on research in green MDD beyond video games.

ACKNOWLEDGEMENTS

Partially supported by MINECO under the Project VARIATIVA (PID2021-128695OB-I00), by UE under the Project OASSIS (PID2021-122554OBC31/AEI/10.13039/ 501100011033/FEDER), by CECD (JCCM) and FEDER funds under the Project EMMA (SBPLY/21/180501/000115), by MCIN/AEI/ 10.13039/501100011033 and European Union NextGenerationEU/ under the project SEEAT (PDC2022-133249-C31), and by by MCIN/AEI/ 10.13039/501100011033 and European Union NextGenerationEU/PRTR under the project PLAGEMIS (TED2021-129245B-C22).

REFERENCES

[1] 2023. Electricity consumption per dwelling. https://www.odyssee-mure.eu/publications/efficiency-by-sector/households/electricity-consumption-dwelling.html. [Online; accessed 16-January-2024].
[2] activeplayer.io Game Statistics Authority. 2023. Fortnite Live Player Count and Statistics. https://activeplayer.io/fortnite/. [Online; accessed 11-December-2023].
[3] Anders Andrae. 2019. Projecting the chiaroscuro of the electricity use of communication and computing from 2018 to 2030. https://doi.org/10.13140/RG.2.2.25103.02724
[4] Ankita Atrey, Nikita Jain, and N Ch Sriman Narayana Iyenger. 2013. A Study on Green Cloud Computing. *International Journal of Grid and Distributed Computing* 6 (12 2013), 93–102. https://doi.org/10.14257/ijgdc.2013.6.6.08
[5] Newzoo International B.V. 2023. How consumers engage with video games today. https://newzoo.com/resources/trend-reports/global-gamer-study-free-report-2023?utm_campaign=2023-06-GGS-GGS%202023%20launch%20report&utm_source=Press. [Online; accessed 16-January-2024].
[6] Newzoo International B.V. 2023. Newzoo's Global Games Market Report 2023. https://newzoo.com/resources/trend-reports/newzoo-global-games-market-report-2023-free-version#:~:text=Highlights%20of%20the%20free%202023%20Global%20Games%20Market%20Report%3A&text=The%20number%20of%20players%20worldwide,year%20growth%20of%20%2B0.6%25.. [Online; accessed 16-January-2024].
[7] Coral Calero, Macario Polo, and Mª Ángeles Moraga. 2021. Investigating the impact on execution time and energy consumption of developing with Spring. *Sustainable Computing: Informatics and Systems* 32 (2021), 100603.
[8] Jessica Clement. 2023. Average weekly hours spent playing video games in selected countries worldwide as of January 2021. https://www.statista.com/statistics/273829/average-game-hours-per-day-of-video-gamers-in-selected-countries/. [Online; accessed 19-December-2023].
[9] Brian Crecente. 2023. Unreal Engine powers stylish stellar mystery The Invincible. https://www.unrealengine.com/en-US/developer-interviews/unreal-engine-powers-stylish-stellar-mystery-the-invincible. [Online; accessed 26-March-2024].
[10] Istvan David and Dominik Bork. 2023. Towards a taxonomy of digital twin evolution for technical sustainability. In *2023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. IEEE, 934–938.
[11] Eleonora Fanouraki. 2022. Did you know that 60% of game developers use game engines? https://www.slashdata.co/blog/did-you-know-that-60-of-game-developers-use-game-engines. [Online; accessed 05-December-2023].
[12] Gabriele Gramelsberger, Hendrik Kausch, Judith Michael, Frank Piller, Ferdinanda Ponci, Aaron Praktiknjo, Bernhard Rumpe, Rega Sota, and Sandra Venghaus. 2023. Enabling informed sustainability decisions: sustainability assessment in iterative system modeling. In *2023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. IEEE, 964–968.
[13] CD PROJEKT Group. 2023. Consolidated Report for Q1 2023. https://www.cdprojekt.com/en/investors/regulatory-announcements/consolidated-report-for-q1-2023/. [Online; accessed 30-March-2024].
[14] Achim Guldner, Rabea Bender, Coral Calero, Giovanni S Fernando, Markus Funke, Jens Gröger, Lorenz M Hilty, Julian Hörnschemeyer, Geerd-Dietger Hoffmann, Dennis Junger, et al. 2024. Development and evaluation of a reference measurement model for assessing the resource and energy efficiency of software products and components—Green Software Measurement Model (GSMM). *Future Generation Computer Systems* (2024).
[15] María Gutiérrez, Ma Ángeles Moraga, Félix García, and Coral Calero. 2023. Green-IN Machine Learning at a Glance. *Computer* 56, 6 (2023), 35–43.
[16] Epic Games Inc. 2023. Real-time round-up: the state of interactive 3D. https://www.unrealengine.com/en-US/blog/real-time-round-up-the-state-of-interactive-3d. [Online; accessed 11-December-2023].
[17] Epic Games Inc. 2024. About Epic Games. https://www.epicgames.com/site/en-US/about. [Online; accessed 13-March-2024].
[18] Epic Games Inc. 2024. Blueprint Compiler Overview. https://dev.epicgames.com/documentation/en-us/unreal-engine/compiler-overview-for-blueprints-visual-scripting-in-unreal-engine?application_version=5.3. [Online; accessed 25-March-2024].
[19] Epic Games Inc. 2024. The most powerful real-time 3D creation tool - Unreal Engine. https://www.unrealengine.com/. [Online; accessed 26-March-2024].
[20] University of Lille Inria. 2023. PowerAPI. https://powerapi.org/. [Online; accessed 27-May-2024].
[21] Daniel Johnson, Ella Horton, Rory Mulcahy, and Marcus Foth. 2017. Gamification and serious games within the domain of domestic energy consumption: A systematic review. *Renewable and Sustainable Energy Reviews* 73 (2017), 249–264.
[22] Jörg Kienzle, Gunter Mussbacher, Benoit Combemale, Lucy Bastin, Nelly Bencomo, Jean-Michel Bruel, Christoph Becker, Stefanie Betz, Ruzanna Chitchyan, Betty HC Cheng, et al. 2020. Toward model-driven sustainability evaluation. *Commun. ACM* 63, 3 (2020), 80–91.
[23] Nupur Kothari and Arka Bhattacharya. 2009. Joulemeter: Virtual machine power measurement and management. *MSR Tech Report* (2009).
[24] Javier Mancebo, Coral Calero, Félix García, Mª Ángeles Moraga, and Ignacio García-Rodríguez de Guzmán. 2021. FEETINGS: Framework for energy efficiency testing to improve environmental goal of the software. *Sustainable Computing: Informatics and Systems* 30 (2021), 100558.
[25] R. Pereira, M. Couto, F. Ribeiro, R. Rua, J. Cunha, J.P. Fernandes, and J. Saraiva. 2021. Ranking programming languages by energy efficiency. *Science of Computer Programming* 205 (2021). https://doi.org/10.1016/j.scico.2021.102609
[26] Carlos Pérez, Ana C Marcén, Javier Verón, and Carlos Cetina. 2023. A survey on green computing in video games: The dawn of Green Video Games. (Dec. 2023). arXiv:2312.09053 [cs.SE]
[27] Carlos Pérez, Javier Verón, Félix García, M Ángeles Moraga, Coral Calero, and Carlos Cetina. 2024. A comparative analysis of energy consumption between the widespread unreal and Unity video game engines. (Feb. 2024). arXiv:2402.06346 [cs.SE]
[28] Clauirton Siebra, Paulo Costa, Rafael Marques, Andre L M Santos, and Fabio Q B Silva. 2011. Towards a green mobile development and certification. In *2011 IEEE 7th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*. 288–294. https://doi.org/10.1109/WiMOB.2011.6085386
[29] Nikolai Baskin Sokolov. 2021. Unreal Engine 5: C++ vs Blueprints. https://gamedev.gg/c-vs-blueprint-in-unreal-5/. [Online; accessed 26-March-2024].
[30] Jimmy Thang. 2019. Solo dev Gwen Frey explains how she developed puzzle game Kine using only Blueprints. https://www.unrealengine.com/en-US/developer-interviews/solo-dev-gwen-frey-explains-how-she-developed-puzzle-game-kine-using-only-blueprints. [Online; accessed 26-March-2024].
[31] Lizhe Wang and Samee Khan. 2011. Review of performance metrics for green data centers: A taxonomy study. *The Journal of Supercomputing* 63 (03 2011), 1–18. https://doi.org/10.1007/s11227-011-0704-3
[32] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. 2012. *Experimentation in Software Engineering*. Springer Science & Business Media.
[33] SlashData ©. 2022. State of the Developer Nation 23rd Edition. https://docsend.com/view/4cmnvxa2xb5jd6hr?utm_source=SlashDataco&utm_medium=FreeReports_SoN23. [Online; accessed 29-March-2024].

accepted 14 June 2024