JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 31 (2024) 68-72
Received: 6 October 2023
Accepted: 25 October 2023

Comparative analysis of the implementation performance using selected scripting languages in the Godot game engine

Analiza porównawcza wydajności implementacji wykorzystującej wybrane języki skryptowe w silniku gier Godot

Sebastian Alchimowicz*, Małgorzata Plechawska-Wójcik

Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract
This article describes a comparative analysis of the implementation performance of selected scripting languages on the Godot game engine. In order to analyze the implementation of scripting languages, research scenarios were designed in which the scripts were written in a similar way to facilitate the analysis of the performance of their implementation. The study took into account parameters such as the execution time of a given script, processor time and the amount of RAM used. Based on the results obtained, averages were determined and presented in charts to facilitate their interpretation. The conducted research allowed for a comparative analysis between scripting languages. The analysis showed that each language is better suited for different types of projects, with GDScript being better for smaller projects and C# for more complex projects.

Keywords: Godot; implementation efficiency; scripting languages

Streszczenie
W niniejszym artykule opisano analizę porównawczą wydajności implementacji wybranych języków skryptowych na silniku gier Godot. W celu przeprowadzenia analizy implementacji języków skryptowych zaprojektowano scenariusze badawcze, w których skrypty zostały napisane w podobny sposób, aby ułatwić analizę wydajności ich implementacji. W badaniu wzięto pod uwagę parametry takie jak czas wykonania danego skryptu, czas pracy procesora oraz ilości wykorzystanej pamięci RAM. Na podstawie otrzymanych wyników wyznaczono średnie, które przedstawiono na wykresach dla ułatwienia ich interpretacji. Przeprowadzone badania pozwoliły na wykonanie analizy porównawczej pomiędzy językami skryptowymi. Analiza wykazała, że każdy z języków nadaje się lepiej do innego rodzaju projektów, w szczególności GDScript lepiej sprawdza się w mniejszych, a C# w bardziej rozbudowanych projektach.

Słowa kluczowe: Godot; wydajność implementacji; języki skryptowe

*Corresponding author
Email address: **sebastian.alchimowicz@pollub.edu.pl** (S. Alchimowicz)

1. Wstęp

Języki skryptowe są jednym z wielu elementów wymaganych do stworzenia działającej gry wideo. W procesie tworzenia gier szczególną uwagę zwraca się na wybór silnika, na którym ma być zbudowany projekt. Wiele z tych silników umożliwia wykorzystanie dedykowanego języka skryptowego lub opcjonalnego języka programowania. W przypadku możliwości wyboru technologii oraz języka programowania tworzonego projektu, decyzja ta zostaje podjęta względem wymagań projektu, preferencji i umiejętności programistów. Najczęściej wybierany zostaje język dedykowany dla danego silnika dlatego, że wokół niego skupiona jest bardziej doświadczona społeczność, a co za tym idzie możliwe jest łatwiejsze znalezienie rozwiązania dla danego problemu. Jednakże przy dokonywaniu wyboru należy także uwzględnić wydajność danego języka skryptowego. Pozwala to programiście na podjęcie decyzji dotyczącej wyboru języka skryptowego do projektu. Poniższa praca ma na celu przeprowadzenie analizy porównawczej wydajności implementacji wybranych języków skryptowych.

Celem artykułu jest przeprowadzenie analizy porównawczej względem wydajności implementacji wybranych języków skryptowych na silniku gier Godot. Badanie zostanie przeprowadzone poprzez napisanie dwóch modułów do projektu, odpowiednio w językach C# i GDScript, a następnie porównanie ich wydajności.

Na potrzeby prowadzonej analizy porównawczej postawiono następujące tezy:

- Język GDScript sprawdzi się lepiej niż C# w środowisku Godot.
- Dla każdego języka skryptowego zużycie pamięci RAM będzie podobne.

2. Przegląd literatury

Wraz z wzrostem popularności gier komputerowych powstało wiele różnych środowisk umożliwiających ich tworzenie. Z tą tematyką jest związanych wiele prac naukowych porównujących wydajności lub analizujące wybrane możliwości środowisk do tworzenia gier [1-11]. W literaturze naukowej można również odnaleźć takie artykuły, w których analiza tych programów jest na podstawie opinii ogólnej pozyskanej przy użyciu ankiet [12 i 13].

W artykule [1] celem prowadzonej analizy było sprawdzenie zachowania wybranych silników przy różnych ustawieniach graficznych. W tym celu przygotowane zostały odpowiednie scenariusze badawcze dla każdego z silników. Praca naukowa [2] zawiera porównanie silników gier pod względem wizualizacji terenu 3D. Przeprowadzona analiza polegała na określaniu dostępnych funkcji środowiska i porównaniu efektów otrzymanych z tworzenia obszaru 3D. Natomiast w artykule [3] autorzy przeprowadzili porównanie środowisk do tworzenia gier pod względem oświetlenia. W kolejnych pozycjach autorzy [4-6] również porównywali ze sobą wybrane silniki gier w celu wyznaczenia, który jest lepszy.

W literaturze naukowej można także znaleźć takie artykuły, które analizują wybrane możliwości silnika gier [7-11]. W artykule [7] przedstawiono wykorzystanie post-processingu dla silnika Unreal Engine 4. Sprawdzony został wpływ poszczególnych ustawień na wydajność danego silnika. W publikacji naukowej [8] wykorzystano środowisko Unity w celu programowania i animowania robotów. W przypadku pozostałej literatury [9-11] również zbadano bardziej szczegółowo wybrane możliwość silnika gier.

Z przeglądu literatury wynika, że mniejszą uwagę zwraca się na wykorzystywany język skryptowy, ponieważ ważniejsze są możliwość danego silnika. Jednak w każdym z artykułów, do prowadzenia analizy, tworzone są scenariusze wykorzystujące język skryptowy. Większość z dostępnych z dostępnych programów nie pozwala na wybranie języka programowania, jednak są wyjątki które dają taką możliwość. Istnieje wiele języków skryptowych posiadających różne zastosowania i cechy, które mogą wpłynąć na jakość i wydajność projektu. Dlatego ważne jest, porównanie języków programowania, aby wybrać najbardziej efektywny dla danego projektu.

3. Metoda badania

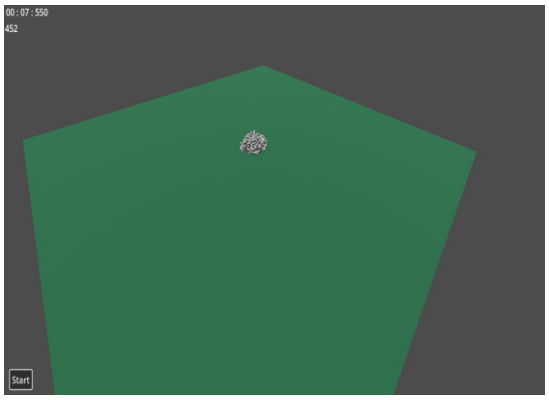
W celu przeprowadzenia analizy implementacji języków skryptowych zaprojektowano scenariusze badawcze mające na celu wyznaczenie czasu wykonania danego skryptu, czasu pracy procesora oraz ilości wykorzystanej pamięci RAM. Z tego powodu oba skrypty zostały napisane w podobny sposób, aby ułatwić analizę wydajności ich implementacji.

3.1. Plan badania

W pierwszym etapie przeprowadzone zostało badanie generowania obiektów, dla którego określono progi w liczbie 500, 750, 1000 i 1250 tworzonych modeli. Pierwsze badanie wykonywane zostało dla języka GDScript, po 7 testów dla każdego z progów liczbowych, wraz z kilkuminutowymi przerwami pomiędzy zmianą wartości tworzonych obiektów. Następnie zrestartowano komputer testowy i zamieniono wykonywany skrypt na język C#, gdzie wykonano badanie podobnie jak dla poprzedniego języka. W podobny sposób przeprowadzono badania dla generowania obszaru dla wymiarów siatek 20x20 (400 kratek), 40x40 (1600 kratek) i 60x60 (3600 kratek). Badania powtórzono po 8 testów w ten sam sposób.

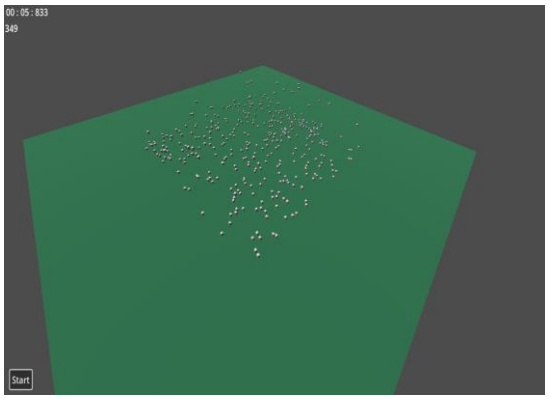
3.2. Generowanie obiektów

Pierwszym scenariuszem badawczym jest analiza wydajności silnika fizyki i renderowania. Początkowym etapem dla tego badania jest wygenerowanie odpowiedniej liczby obiektów (sześcianów), które posiadają właściwości umożliwiające symulowanie fizyki. Obiekty zostaną wygenerowane w dwóch sytuacjach badawczych – bez (Rysunek 1) oraz z (Rysunek 2) wykorzystaniem skryptu rozmieszczenia obiektów. W tym celu stworzono jeden scenariusz, który obejmuje oba przypadki. W przypadku pierwszym obiekty są tworzone w pozycji startowej, natomiast w drugim pojawiają się w losowej pozycji w trakcie doświadczenia.



Rysunek 1: Generowania obiektów bez rozmieszczenia obiektów w trakcie działania.

Dla pierwszego przypadku obiekty tworzone są w pozycji początkowej i nowo stworzony obiekt przesuwa pozostałe wygenerowane modele, ponieważ posiada on fizykę co przyczynia się do obliczania nadmiernej kolizji obiektów.

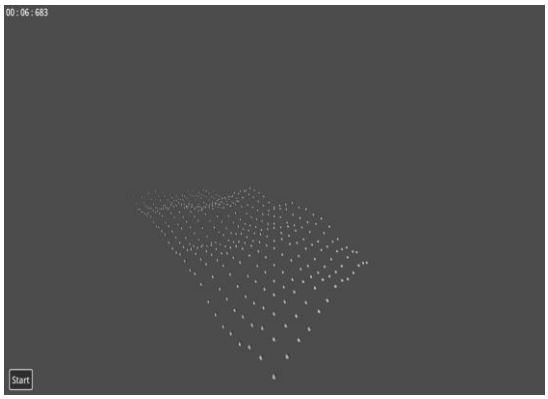


Rysunek 2: Generowania obiektów z losowym rozmieszczeniem obiektów w trakcie działania.

W drugim przypadku przy losowym rozstawieniu tych obiektów nie występuje nadmierna kolizja pomiędzy tymi modelami.

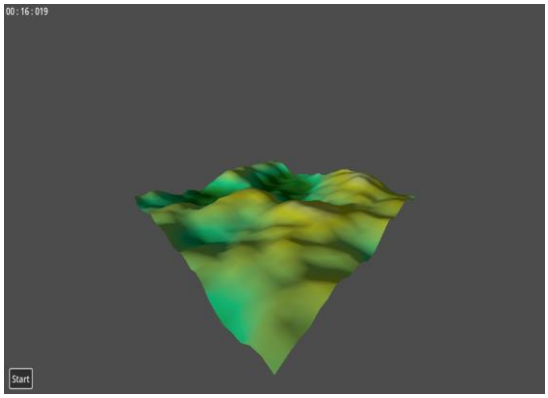
3.3. Generowanie obszaru

Kolejnym scenariuszem badawczym jest wygenerowanie obszaru poprzez rozmieszczenie kul w ustalonej od siebie odległości i na różnej wysokości. Ma to na celu stworzenia zbioru punktów, który posłuży do wygenerowania trójwymiarowej siatki (Rysunek 3).



Rysunek 3: Generowania obszaru w trakcie wykonywania badania.

Kolejnym krokiem w tworzeniu obszaru po rozmieszczeniu jest utworzenie struktury. Wykonywane są w tym celu odpowiednie instrukcje warunkowe, następnie po ich zakończeniu użyta jest metoda zwracająca wynikowy obiekt siatki. Na koniec obiekt zostaje przypisany do właściwości sceny, co wywoła wygenerowanie modelu siatki. W końcowym etapie nastąpi oczyszczenie sceny z wcześniej rozmieszczonych obiektów, wraz z aktualizacją mapy cieniowania tworzonego obszaru (Rysunek 4).



Rysunek 4: Zakończenie wykonywania generowania obszaru.

3.4. Sprzęt użyty do badania

Do przeprowadzania badań wykorzystano komputer stacjonarny wyposażony w płytę główna B450 AORUS Elite V2 z procesorem AMD Ryzen 7 5700G 8 rdzeni i 16 wątków Zen 3 zintegrowany z grafiką Radeon Vega 8, pamięcią GOODRAM 16GB DDR4-3000MHz i dyskiem GOODRAM 256GB 2,5" SATA SSD CX400. Na komputerze zainstalowano system Arch Linux wraz z środowiskiem graficznym xfce.

4. Wyniki

Na podstawie otrzymanej dużej liczby wyników z przeprowadzonych badań wyznaczono średnie, które przedstawiono na wykresach dla ułatwienia ich interpretacji.

4.1. Wyniki generowania obiektów

Dla średnich wyników generowania obiektów bez rozmieszczenia różnica jest widoczna przy mniejszej liczbie modeli, takich jak 500 i 750. Natomiast wraz z wzrostem liczby tworzonych modeli jest zauważalny wzrost czasu wykonania i pracy procesora (Tabela 1 i 2).

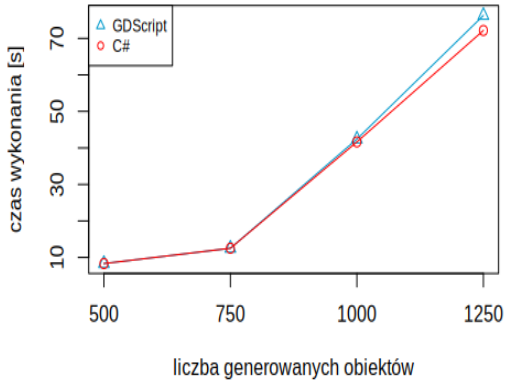
Tabela 1: Średnie wyników generowania obiektów bez rozmieszczenia dla GDScript

Liczba	Time [s:ms]	CPU [ms]	RAM [MiB]
500	8:333	165.4	254.7
750	12:517	277.2	266.6
1000	42:422	1160	278.5
1250	76:279	1310	280.9

Tabela 2: Średnie wyników generowania obiektów bez rozmieszczenia dla C#

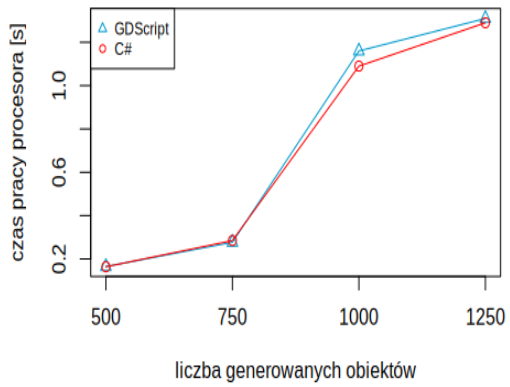
Liczba	Time [s:ms]	CPU [ms]	RAM [MiB]
500	8:333	164.2	256.5
750	12:527	285.6	267.9
1000	41:642	1090	278.5
1250	72:164	1290	280.9

Średnie wyników czas wykonania badania dla 500 i 750 obiektów pokrywają się, dla 1000 generowanych obiektów różnica zaczyna być widoczna i powiększa się dla badania generowania 1250 obiektów.



Rysunek 5: Wykres średniej czasów wykonania generowania obiektów bez rozmieszczenia.

Podobne wyniki otrzymano dla średniego czasu pracy procesora, gdzie dla 500 i 750 obiektów wyniki pokrywają się. Przy 1000 obiektach język GDScript osiąga gorsze wyniki niż C#, natomiast przy 1250 wyniki są niemal identyczne (Rysunek 6). Natomiast różnica zajętej pamięci RAM pomiędzy językami niewielka.



Rysunek 6: Wykres średniej czasów pracy procesora generowania obiektów bez rozmieszczenia.

Z kolei w przypadku generowania z losowym rozmieszczeniem obciążenie nadmiarem obliczeń kolizji jest bardzo niewielkie. Z tego powodu czasy wykonania dla obydwu języków są podobne, różnice można jednak zauważyć w czasie pracy procesora (Tabela 3 i 4).

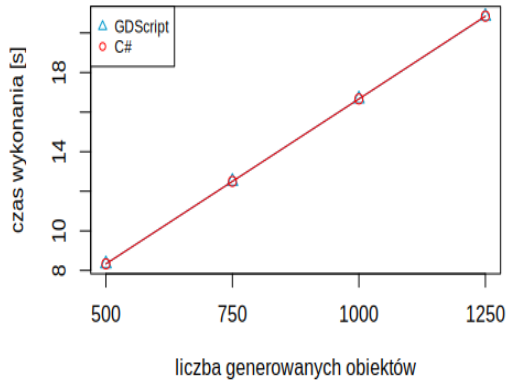
Tabela 3: Średnie wyników generowania obiektów z losowym rozmieszczeniem dla GDScript

Liczba	Time [s:ms]	CPU [ms]	RAM [MiB]
500	8:333	33	257.1
750	12:500	38.4	269.1
1000	16:666	46.9	272.4
1250	20:836	69.3	283.2

Tabela 4 wyników generowania obiektów z losowym rozmieszczeniem dla C#

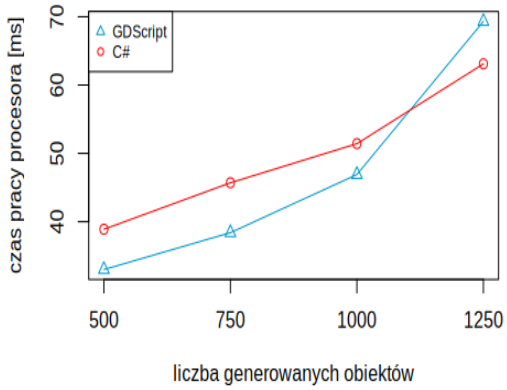
Liczba	Time [s:ms]	CPU [ms]	RAM [MiB]
500	8:333	38.9	257.6
750	12:500	45.7	269.2
1000	16:666	51.42	272.4
1250	20:849	63.1	283.7

W przypadku generowania z rozmieszczeniem na wykresie średnich czasów wykonania badania wyniki pokrywają się. Jedynie dla wyników przy 1250 obiektach widoczna jest niewielka różnica, wyrażana w milisekundach (Rysunek 7).



Rysunek 7: Wykres średniej czasów wykonania generowania obiektów z losowym rozmieszczeniem.

Dla czasu pracy procesora dla 500 i 750 generowanych obiektów wyniki języka GDScript są mniejsze od C#. Gdy liczba generowanych obiektów osiąga 1000, czas ten zaczyna zbliżać się dla obydwu języków. Jednakże dla większej liczby generowanych obiektów, takiej jak 1250, wyniki czasu pracy procesora dla C# są mniejsze (Rysunek 8).



Rysunek 8: Wykres średniej czasów pracy procesora generowania obiektów z losowym rozmieszczeniem.

Należy zaznaczyć, że różnica w zajętej pamięci RAM pomiędzy językami dla generowania obiektów z losowym rozmieszczeniem jest niewielka.

4.2. Wyniki generowania obszaru

W przypadku generowania obszaru wzrost szerokości siatki przyczynił się do znaczengo wzrostu czasu wykonania i pracy procesora (Tabela 5 i 6).

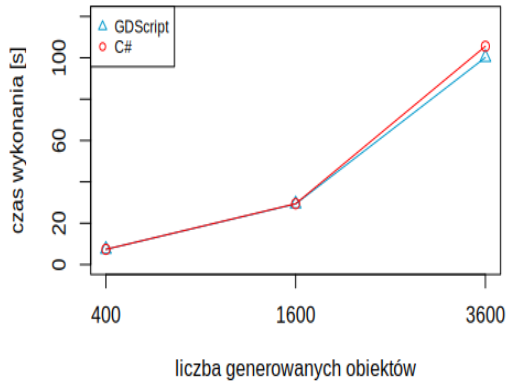
Tabela 5: Średnie wyników generowania oszbaru dla GDScript

Liczba	Time [s:ms]	CPU [ms]	RAM [MiB]
400	7:350	26.1	257.3
1600	29:306	97.9	286.8
3600	100:162	124.7	316.7

Tabela 6: Średnie wyników generowania oszbaru dla C#

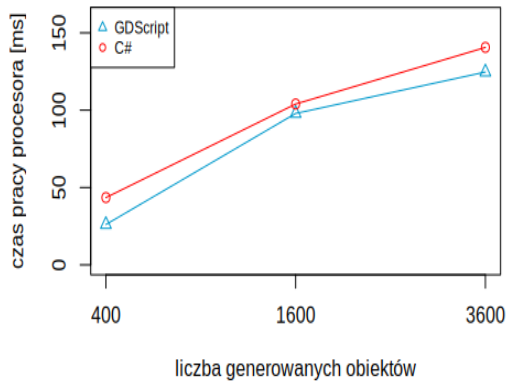
Liczba	Time [s:ms]	CPU [ms]	RAM [MiB]
400	7:355	43.5	256.5
1600	29:429	104	287.2
3600	105:575	140.6	316.3

Dla generowanego obszaru o szerokości 400 i 1600 kratek różnice pomiędzy czasami wykonania badania dla obu języków nie są zauważalne, różnica wyrażana jest w milisekundach. Jednakże już przy obszarze o szerokości 3600 kratek, skrypt w języku GDScript wygenerował obszar szybciej niż skrypt w języku C# (Rysunek 9).



Rysunek 9: Wykres średniej czasów wykonania generowania obszaru.

Podobną sytuację można zauważyć w przypadku czasu pracy procesora, gdzie średnie wyniki dla języka GDScript są mniejsze niż wyniki C#. Wzrost jest widoczny przy obszarze o szerokości 1600 kratek, gdzie GDScript osiągnął wyniki zbliżone do C# (Rysunek 10). Należy zaznaczyć, że różnica w zajętej pamięci RAM pomiędzy językami jest niewielka.



Rysunek 10: Wykres średniej czasów pracy procesora generowania obiektu.

5. Wnioski

Przeprowadzona analiza pozwoliła stwierdzić, że każdy z języków ma swoje mocne i słabe strony. Język GDScript okazał się prostszy w zrozumieniu i bardziej przejrzysty w kodzie, co sprawia, że jest łatwiejszy do nauczenia w przeciwieństwie do C#. Dlatego sprawdził się on dobrze dla mniejszej liczby generowanych obiektów, co jest zauważalne przy generowaniu obiektów z rozmieszczeniem. Również język GDScript lepiej poradził sobie z generowaniem obszaru. Natomiast język C# wykazał się lepszą obsługą większych projektów, zwłaszcza przy nadmiernej kolizji obiektów. Na podstawie przedstawionych wyników z badań można stwierdzić, że pierwsza teza jest nieprawdziwa.

Wykorzystana pamięć RAM wzrastała przy każdym kolejnym prowadzonym teście z powodu większej liczby obiektów dodanych w trakcie testów, ale nie miało to większego wpływu na różnicę pomiędzy wybranymi językami, co potwierdza drugą tezę.

Literatura

[1] W. Szelug, Analiza porównawcza wydajności silników Flax engine i Unity, JCSI 25 (2022) 358-361.

[2] R. Ch. Mat, A. R. M. Shariff, A. N. Zulkifli, M. S. M. Rahim, M. H. Mahayudin, Using game engine for 3D terrain visualisation of GIS data: A review, IOP Conference Series: Earth and Environmental Science 20 (2014) 012037.

[3] C. Lambru, A. Morar, F. Moldoveanu, V. Asavei, A. Moldoveanu, Comparative Analysis of Real-Time Global Illumination Techniques in Current Game Engines, IEEE Access 9 (2021) 125158-125183.

[4] P. E. Dickson, J. E. Block, G. N. Echevarria, K. C. Keenan An Experience-based Comparison of Unity and Unreal fora Stand-alone 3D Game Development Course, ItiCSE 17 (2017) 70-75.

[5] H. Żukowski, Porównanie wydajności trójwymiarowych gier z użyciem silników CryEngine i Unity, Praca magisterska, Politechnika Lubelska, Lublin, 2019.

[6] J. Tomalá-Gonzáles, J. Guamán-Quinche, E. Guamán-Quinche, W. Chamba-Zaragocin, S. Mendoza-Betancourt, Serious Games: Review of methodologies and Games engines for their development, Iberian Conference on Information Systems and Technologies (CISTI) 15 (2020) 24-27.

[7] E. Puławski, M. Tokarski, Wykorzystanie postprocesingu i jego wpływu na wydajność renderowania w silniku Unreal Engine 4, JCSI 10 (2019) 54-61.

[8] C. Bartneck, M. Soucy, K. Fleuret, E. B. Sandoval, The robot engine — Making the unity 3D game engine work for HRI, IEEE International Symposium on Robot and Human Interactive Communication 24 (2015) 431-437.

[9] X. Wu, Q. Fu, Science Unreal Domed Screen Film-making and Application Based on Unreal Engine Technology, Proceedings of the 3rd Asia-Pacific Conference on Image Processing, Electronics and Computers (2022) 206-210.

[10] G. Gabajová, M. Krajčovič, M. Matys, B. Furmannová, N. Burganová, Designing Virtual Workplace using Unity 3D Game Engine, Acta Tecnología Vol. 7 No. 1 (2021) 35–39.

[11] A. Hussain, H. Shakeel, F. Hussain, N. Uddin, T. L. Ghouri, Unity Game Development Engine: A Technical Survey, USJICT Vol. 4 No. 2 (2020) 73-81.

[12] X. Christopoulou, S. Xinogalos, Overview and Comparative Analysis of Game Engines for Desktop and Mobile Devices, International Journal of Serious Games Vol. 4 No. 4 (2017) 21-36.

[13] B. Cowan, B. Kapralos, A Survey of Frameworks and Games Engines for Serious Game Development, IEEE International Conference on Advanced Learning Technologies 14 (2014) 662-664.