JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 37 (2025) 431–435
Received: 19 June 2025
Accepted: 18 October 2025

Analysis of performance optimization methods for 3D games in the Unity environment

Analiza metod optymalizacji wydajności dla gier 3D w środowisku Unity

Maciej Potręć*, Marcin Badurowicz

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

Abstract
The paper analyzes methods for optimizing performance in 3D games created in Unity. Based on the literature, the following techniques are discussed: GPU Instancing, Static Batching, Occlusion Culling, Level of Detail (LOD), Unity Jobs, Object Pooling, and the tick system. In four test scenarios, metrics (FPS, RAM, batch count) were collected using Unity Profiler. The results showed that GPU Instancing and Static Batching increase FPS (10,9 % and 20,4 %) and reduce memory usage, while LOD increases FPS by over 380 % with a minimal increase in RAM. Occlusion Culling is only effective with large objects. Unity Jobs, Object Pooling, and ticks improve performance by 14-27 %. The choice of methods should depend on the nature of the scene. Further research on other platforms is recommended.

Keywords: performance optimization; 3D games; Unity; FPS

Streszczenie
Praca analizuje metody optymalizacji wydajności w grach 3D tworzonych w Unity. Na podstawie literatury omówiono techniki: GPU Instancing, Static Batching, Occlusion Culling, Level of Detail (LOD), Unity Jobs, Object Pooling i system ticków. W czterech scenariuszach testowych zbierano metryki (FPS, RAM, batch count) za pomocą Unity Profiler. Wyniki wykazały, że GPU Instancing i Static Batching podnoszą FPS (10,9 % i 20,4 %) i redukują wykorzystanie pamięci, a LOD zwiększa FPS o ponad 380 % przy minimalnym wzroście RAM. Occlusion Culling działa skutecznie tylko przy dużych obiektach. Unity Jobs, Object Pooling i ticki poprawiają wydajność o 14-27 %. Dobór metod powinien zależeć od charakteru sceny. Dalsze badania na innych platformach zalecane.

Słowa kluczowe: optymalizacja wydajności; gry 3D; Unity; FPS

*Corresponding author
Email address: s95536@pollub.edu.pl (M. Potręć)

1. Wstęp

W ostatnich latach branża gier komputerowych rozwija się w zawrotnym tempie. Tylko w 2024 roku wartość globalnego rynku gier osiągnęła imponujące 187,7 miliarda dolarów, co oznacza wzrost o 2,1% względem roku poprzedniego [1]. Równocześnie liczba aktywnych graczy na całym świecie w 2025 roku sięgnęła aż 3,32 miliarda, z czego ponad 1,5 miliarda jest skłonnych płacić za gry oraz związane z nimi usługi.

Tak dynamiczny rozwój nie pozostaje bez wpływu na oczekiwania graczy - ci coraz częściej domagają się nie tylko realistycznej oprawy graficznej, ale i perfekcyjnej płynności rozgrywki. Nowoczesne produkcje sięgają po coraz bardziej zaawansowane technologie, które pozwalają zanurzyć się w świecie gry na niespotykaną wcześniej skalę. Aby jednak sprostać tym rosnącym wymaganiom, twórcy muszą sięgać po skuteczne techniki optymalizacji zarówno kodu, jak i grafiki.

Jednym z kluczowych elementów wpływających na komfort gracza jest wydajność gry, najczęściej oceniana poprzez liczbę klatek na sekundę (FPS). Wraz ze wzrostem złożoności nowych tytułów rośnie też znaczenie problematyki optymalizacji. Szczególnie istotne staje się zapewnienie, by także użytkownicy korzystający ze słabszego sprzętu mogli cieszyć się płynną, niezakłóconą rozgrywką.

2. Przegląd literatury

Twierdzenie, że wydajność gier komputerowych jest jednym z kluczowych czynników wpływających zarówno na doświadczenia, jak i na wyniki osiągane przez graczy, znajduje potwierdzenie w licznych publikacjach. W pracy [2] wykazano, że wyższa liczba klatek na sekundę przekłada się na lepsze doświadczenia graczy. W pracy [3] wykazano, że liczba klatek na sekundę znacząco wpływa na efektywność graczy w strzelankach FPS (First Person Shooter). W pracy [4] przeanalizowano zachowania graczy e-sporcie czyli w kompetytywnym graniu w gry komputerowe [5]. Autorzy wykazali, że gracze często obniżają jakość grafiki w celu poprawienia płynności gier. Takie praktyki obrazują kluczową rolę procesu optymalizacji gier, gdzie płynność rozgrywki może decydować o wynikach rywalizacji. W artykule [6] przeprowadzono analizę i ponowną implementacje mobilnej gry edukacyjnej. Wyniki wcześniejszych badań wykazały problemy z wydajnością, takie jak duży rozmiar pliku gry, opóźnienia między scenami oraz niewystarczająca satysfakcja użytkowników. Autorzy zastosowali techniki optymalizacyjne, koncentrując się na poprawie wydajności oraz zadowolenia użytkowników. Poprawki

obejmowały zmniejszenie rozmiaru plików, optymalizację czasu ładowania oraz poprawę interfejsu użytkownika. Badanie wykazało wzrost zadowolenia użytkowników i wyższą wydajność gry, co potwierdza skuteczność zastosowanych metod. W pracy [7] omówiono problemy optymalizacji scen w grach 3D na urządzeniach mobilnych z używając do tego silnika Unity. W pracy przedstawiono techniki i umiejętności związane z optymalizacją wydajności na telefonach komórkowych, podkreślając kluczowe wyzwania podczas tego procesu. Badania wskazują na rosnące znaczenie gier 3D na urządzeniach mobilnych oraz konieczność stosowania dedykowanych metod optymalizacyjnych w celu dostosowania wydajności do ograniczeń sprzętowych. W pracy [8] podkreślono, że wraz z rozwojem technologii 3D skanowania oraz wzrostem złożoności symulacji komputerowych, zarządzanie poziomem szczegółowości staje się kluczowe. Automatyczne uproszczenie modeli i techniki zarządzania Level of Details (LOD) na czas rzeczywisty pomagają złagodzić przeciążenia współczesnych układów graficznych. W pracy zorganizowano przegląd metod LOD, omawiając m.in. ramy zarządzania LOD, modele uproszczeń i metryki oceny błędów. Autorzy wskazali, że zastosowanie tych technik jest niezbędne dla utrzymania wysokiej wydajności aplikacji korzystających z ogromnych zestawów danych geometrycznych. Podsumowując badania w temacie metod optymalizacji wydajności wskazują na wzrost wydajności aplikacji po ich zastosowaniu.

3. Cel i zakres pracy

Celem tej pracy jest przeprowadzenie analizy wybranych metod optymalizacji wydajności w grach komputerowych, tworzonych przy użyciu środowiska Unity. W niniejszej pracy szczególną uwagę położono na zbadanie wpływu badanych technik na zwiększenie liczby klatek na sekundę podczas rozgrywki.
Zakres pracy obejmuje następujące elementy:

- Analiza istniejących badań naukowych oraz publikacji dotyczących tematu gier komputerowych oraz ich optymalizacji.
- Opis technologii i narzędzi zastosowanych w badaniach.
- Merytoryczny opis metod optymalizacyjny oraz pokazanie ich implementacji w projekcie.
- Opis stanowiska badawczego, metodyki badań oraz scenariuszy testowych.
- Przeprowadzenie testów i zebranie danych do analizy.
- Badanie wpływu różnych metod optymalizacji na wydajność gier w kontekście liczby klatek na sekundę.
- Analiza wyników oraz sformułowanie wniosków.

W ramach pracy sformułowano hipotezy badawcze:

- H1: Gry z zaimplementowanymi metodami optymalizacji wydajności mają większą liczbę klatek na sekundę niż gry bez zaimplementowanych metod.
- H2: Zaimplementowanie metody optymalizacyjnej nie zwiększa zużycia pamięci RAM.

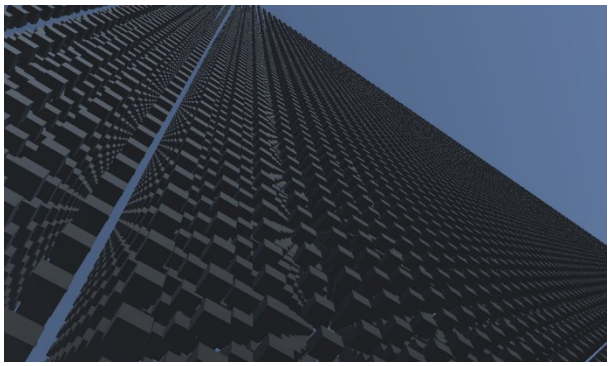
4. Metodyka badawcza

4.1. Opis scenariuszy testowych

W celu porównania wybranych metod optymalizacyjnych w kontekście wydajności stworzono środowisko na podstawie gotowego rozwiązania oferowanego przez Unity, gdzie potokiem renderowania została wybrana wersja Universal Render Pipeline (URP). Ponieważ niektórych metod nie można bezpośrednio porównywać, opracowano cztery scenariusze testowe.
Scenariusz 1. Polega na poruszaniu kamerą wokół sceny, w której umieszczono 100 000 identycznych obiektów. Celem jest sprawdzenie wpływu dużej liczby statycznych instancji na wydajność renderowania oraz liczbę klatek na sekundę. Przykładową scenę zaprezentowano na Rysunku 1.

Rysunek 1: Przykładowe zdjęcie prezentujące scenę z scenariusza 1.

Scenariusz 2. Zakłada poruszanie się postaci po rozległej mapie, na której rozmieszczono liczne drobne elementy, takie jak drzewa, rośliny oraz kamienie. Test ten ma uwidocznić, jak obecność różnorodnych obiektów wpływa na płynność rozgrywki. Przykładową scenę zaprezentowano na Rysunku 2.



Rysunek 2: Przykładowe zdjęcie prezentujące scenę z scenariusza 2.

Scenariusz 3. Symulowana rozgrywka w stylu „obrony wieży”. Na scenie generowani są liczni przeciwnicy, których zadaniem jest zlokalizowanie gracza i poruszanie się w jego kierunku. Gdy przeciwnik zbliży się do określonej odległości od gracza, zostaje usunięty, a następnie odtworzony w losowej pozycji oddalonej od postaci gracza. Dodatkowo dla postaci gracza zaimplementowano mechanizm automatycznego strzelania do najbliższych przeciwników - trafienie skutkuje usunięciem przeciw-

nika i jego ponownym odtworzeniem z dala od gracza. Przykładową scenę zaprezentowano na Rysunku 3.
Scenariusz 4. Polega na obserwacji miasta, w którym znajduje się duża liczba zróżnicowanych modeli 3D. W porównaniu z poprzednimi scenariuszami zastosowano tutaj obiekty, które swoją wielkością zasłaniają niektóre mniejsze elementy sceny. Przykładową scenę zaprezentowano na Rysunku 4.

Rysunek 3: Przykładowe zdjęcie prezentujące scenę z scenariusza 3.

Rysunek 4: Przykładowe zdjęcie prezentujące scenę z scenariusza 4.

4.2. Parametry metod optymalizacyjnych i scen testowych

W celu zapewnienia powtarzalności badań oraz lepszego zrozumienia kontekstu uzyskanych wyników w niniejszym podrozdziale przedstawiono szczegółowe parametry zastosowane dla poszczególnych metod optymalizacyjnych oraz scen testowych. Dobór wartości był podyktowany zarówno ograniczeniami technologicznymi, jak i chęcią uzyskania miarodajnych wyników możliwych do porównania z innymi badaniami.

- W scenariuszu 1 umieszczono 100 000 identycznych obiektów, co pozwoliło zweryfikować efektywność technik w przypadku dużej liczby powtarzalnych elementów. Liczba ta została dobrana tak, aby obciążenie renderowania było znaczące, lecz jednocześnie możliwe do obsłużenia przez sprzęt badawczy.
- Dla Occlusion Culling zastosowano w scenach obejmujących zarówno dużą liczbę drobnych elementów, jak i obiekty o większych gabarytach. Pozwoliło to zweryfikować wpływ parametru minimalnej wielkości obiektu, uwzględnianego podczas procesu cullingu, na efektywność tej techniki. W związku z tym parametr ten został ustawiony na wartość 0,25 dla Scenariusza 2 oraz na 2,5 dla Scenariusza 3.
- Dla LOD w obiektach scenariuszu 2 zdefiniowano trzy poziomy szczegółowości modeli 100 %, 62 %, 32 %. Dodatkowo dla tych poziomów zdefiniowano progi wielkości obiektu względem ekranu dla których będą się zmieniać 25 %, 12,5 %, 1 %. Po 1 % obiekt zostaje wyłączony i jest on niewidoczny dla użytkownika. Wartości te zostały przyjęte na podstawie standardowych praktyk w grach 3D, umożliwiając redukcję liczby wierzchołków i trójkątów w zależności od odległości kamery od obiektu.
- Dla Unity Jobs w scenariuszu 3 zastosowano system wielowątkowego przetwarzania logiki ruchu przeciwników. Liczbę generowanych jednostek ustalono na poziomie 500, co pozwoliło zaobserwować wpływ tej techniki na rozkład obciążenia CPU.
- Dla Object Pooling w scenariuszu 3 liczba przeciwników była utrzymywana na stałym poziomie, a zamiast tworzenia i niszczenia obiektów zastosowano ich ponowne wykorzystanie. Parametr rozmiaru puli został dobrany na podstawie maksymalnej liczby aktywnych jednostek w danym scenariuszu.
- Dla System ticków ustalono krok czasowy odpowiadający 10 operacji wciągu 1s, co jest standardową wartością w grach czasu rzeczywistego. Wartość ta pozwalała na płynne odwzorowanie rozgrywki przy jednoczesnym ograniczeniu liczby obliczeń wykonywanych w krótkim czasie.

Przyjęte parametry zostały dobrane tak, aby z jednej strony umożliwić rzetelne porównanie metod, a z drugiej aby ich wartości były reprezentatywne dla typowych projektów tworzonych w środowisku Unity.

5. Przeprowadzenie badań

Do przeprowadzenia badań wykorzystano środowisko, którego konfigurację przedstawiono w Tabeli 1. Każdy scenariusz przeprowadzono w czterech niezależnych próbach, a w dalszej analizie wykorzystano średnią arytmetyczną uzyskanych pomiarów.

Tabela 1: Zestawienie środowiska

Komponenty	Szczegóły
CPU	AMD Ryzen 7 5800X
GPU	NVIDIA GeForce RTX 2070
RAM	32 GB RAM DDR4 3000MHz
Dysk	NVMe Lexar NM790
System operacyjny	Windows 11
Unity	Unity 6000.0.38f

Aby badania były jak najbardziej powtarzalne, opracowano skrypt nagrywający dane dotyczące ruchu kamery. Dane te umożliwiały późniejsze odtworzenie dokładnego toru ruchu kamery w każdym scenariuszu testowym. Do gromadzenia danych opracowano skrypt, który za pomocą modułu Unity Profiler zbierał metryki w tempie przekraczającym 50 próbek na sekundę czasu

trwania każdego scenariusza. W Tabeli 2 przedstawiono metryki jakie były zbierane.

Tabela 2: Zestaw metryk zbieranych w trakcie testów

Metryka	Opis
FPS	Liczba klatek renderowania przez silnik gry podczas jednej sekundy.
RAM (GB)	Pamięć RAM użyta przez grę.
Batch	Grupa operacji draw call polegających na wysyłaniu informacji z CPU do GPU w celu wyświetlenia obiektu w grze.
Liczba trójkątów	Liczba trójkątów wyrenderowanych przez silnik unity pokazujący złożoność aktualnie testowanej sceny.
Liczba wierzchołków	Liczba wierzchołków pokazująca poziom szczegółowości modeli 3D znajdujących się na scenie podczas jej testowania.

6. Wyniki badań

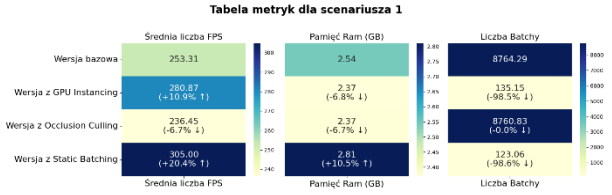
6.1. Wyniki badań dla scenariusza 1

Na Rysunku 5 można zaobserwować, że zastosowanie metody GPU Instancing spowodowało wzrost liczby FPS o 10,9 %, redukcję zużycia pamięci o 6,8 % oraz spadek liczby wywołań grupy renderującej (batch count) aż o 98,5 %.

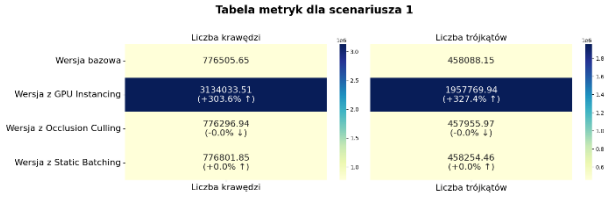
Z kolei zastosowanie metody Occlusion Culling wiązało się ze spadkiem liczby FPS o 6,7 %, redukcją zużycia pamięci o 6,7 %, przy czym liczba wywołań batch count pozostała niezmieniona.

Natomiast Static Batching przyczynił się do wzrostu liczby FPS o 20,4 %, równoczesnego wzrostu zużycia pamięci o 10,5 % oraz spadku liczby wywołań batch count aż o 98,6 %.

Na Rysunku 6 można zaobserwować że liczba trójkątów oraz krawędzi wzrosła o ponad 303,6 % dla krawędzi oraz 327,4 % dla trójkątów względem innych scen. Wynika to z faktu że każda instancja modelu jest liczona osobno w statystykach jednakże w rzeczywistości nie oznacza to większej złożoności obliczeniowej.



Rysunek 5: Wartości zmierzonych metryk dla scenariusza 1.



Rysunek 6: Wartości zmierzonych metryk dla scenariusza 1.

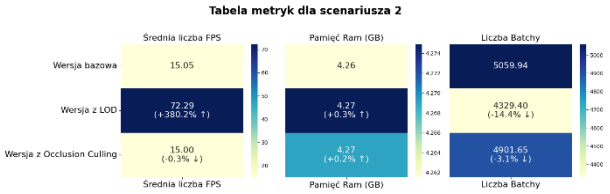
6.2. Wyniki badań dla scenariusza 2

Na Rysunku 7 można zaobserwować, że zastosowanie LOD spowodowało, że liczba FPS wzrosła do 380,2 % wartości wyjściowej, zużycie pamięci zwiększyło się jedynie o 0,3 %, a liczba wywołań batch count zmniejszyła się o 14,4 %.

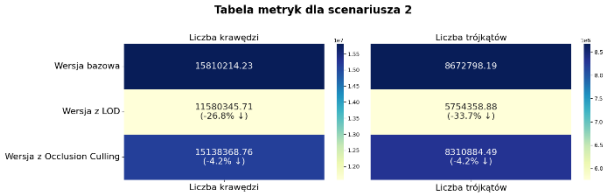
W przypadku zastosowania Occlusion Culling odnotowano spadek liczby FPS o 0,3 %, wzrost zużycia pamięci o 0,2 % oraz zmniejszenie liczby wywołań batch count o 3,1 %.

Na Rysunku 8 można zaobserwować iż LOD zmniejszyło liczbę krawędzi oraz trójkątów o odpowiednio 26,8 % oraz 33,7 %, potwierdza to działanie tej optymalizacji.

Natomiast Occlusion Culling nieznacznie zmniejszył liczbę krawędzi oraz trójkątów o 4,2 % co nie przełożyło się na znaczne zwiększenie wydajności.



Rysunek 7: Wartości zmierzonych metryk dla scenariusza 2.



Rysunek 8: Wartości zmierzonych metryk dla scenariusza 2.

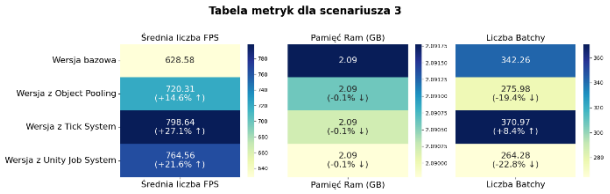
6.3. Wyniki badań dla scenariusza 3

Na Rysunku 9 można zaobserwować, że wykorzystanie systemu Unity Jobs przyczyniło się do wzrostu liczby FPS o 21,6 %, redukcji zużycia pamięci o 0,1 % oraz spadku liczby wywołań batch count o 22,8 %.

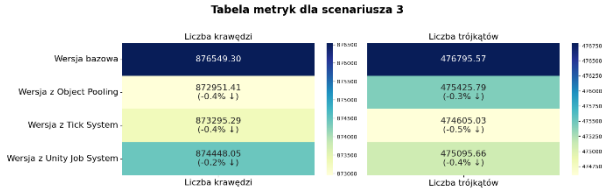
Object Pooling spowodował wzrost liczby FPS o 14,6 %, zmniejszenie zużycia pamięci o 0,1 % oraz spadek liczby wywołań batch count o 19,4 %.

Zastosowanie systemu ticków (metody podziału czasu gry na dyskretne kroki) zwiększyło liczbę FPS o 27,1 %, zmniejszyło zużycie pamięci o 0,1 % oraz spowodowało spadek liczby wywołań batch count o 8,4 %.

Na Rysunku 10 można zaobserwować że dla Object Pooling liczba krawędzi oraz trójkątów zmniejsza się od około 0,2 % a 0,4 %.



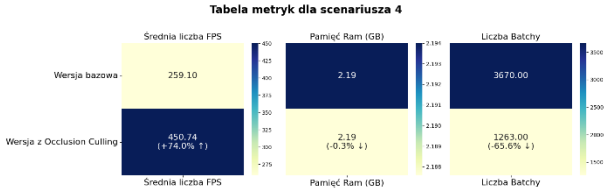
Rysunek 9: Wartości zmierzonych metryk dla scenariusza 3.



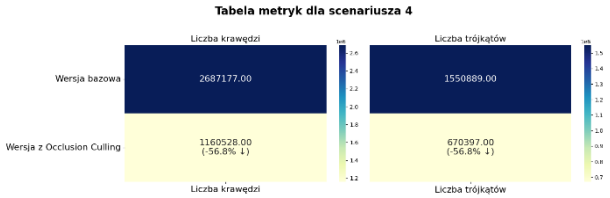
Rysunek 10: Wartości zmierzonych metryk dla scenariusza 3.

6.4. Wyniki badań dla scenariusza 4

Na Rysunku 11 można zaobserwować, że zastosowanie metody Occlusion Culling przyczyniło się do wzrostu liczby FPS o 74 %, redukcji zużycia pamięci o 0,3 % oraz spadku liczby wywołań batch count o 65,6 %. Na rysunku 12 widoczny jest spadek liczby krawędzi oraz trójkątów o 56,8 %.



Rysunek 11: Wartości zmierzonych metryk dla scenariusza 4.



Rysunek 12: Wartości zmierzonych metryk dla scenariusza 4.

7. Wnioski

Wyniki badań pokazują, że zastosowanie metod optymalizacyjnych w większości przypadków poprawiło wydajność gry, co znajduje potwierdzenie w wynikach innych autorów omawianych w przeglądzie literatury [6,7,8].

Natomiast w przypadku Occlusion Culling wyniki są niejednoznaczne, ponieważ w scenariuszach, gdzie w scenie znajduje się wiele małych elementów, koszty zastosowania tej metody przewyższają zyski, doskonale jest to widoczne na rysunku 8 gdzie liczba krawędzi oraz trójkątów zmniejszyła się tylko o 4,2 %. W scenach z dużymi obiektami technika ta przynosi istotne korzyści, zwiększając liczbę klatek na sekundę, a jednocześnie zmniejszając zużycie pamięci RAM dodatkowo zauważalne jest zmniejszenie liczby krawędzi i trójkątów, co obrazuje rysunek 12.

Największy względny wzrost liczby FPS odnotowano w przypadku metody LOD, przy czym zużycie pamięci wzrosło niemal nieznacznie (w granicach typowego odchylenia). Pozostałe techniki zwiększały wydajność w zakresie od około 10 % do około 27 %.

Jeśli chodzi o zużycie pamięci, metody GPU Instancing oraz Occlusion Culling wykazały, że w scenariuszu z wieloma identycznymi obiektami można zaobserwować redukcję wykorzystania pamięci o około 6,7 %, podczas gdy w pozostałych scenariuszach różnice te są nieznaczne i mieszczą się w granicach odchyleń standardowych. Wyniki z przeprowadzonych scenariuszy pokazują, że żadna z hipotez nie została w pełni potwierdzona. W hipotezie H1 wiele metod faktycznie zwiększyło liczbę FPS, jednak zastosowanie Occlusion Culling w tym przypadku spowodowało jej spadek, co zaprzecza sformułowaniu hipotezy. W hipotezie H2 również zaobserwowano wzrost zużycia pamięci w wyniku zastosowania Static Batching, co przeczy założeniu tej hipotezy.

W przyszłości warto przetestować te metody także na innych platformach niż PC/Windows, na przykład na urządzeniach mobilnych oraz konsolach nowej generacji. Dodatkowo niektóre z technik można ze sobą łączyć, dzięki czemu można sprawdzić, które kombinacje przynoszą najlepsze rezultaty wydajnościowe. Innym darmowym silnikiem do tworzenia gier jest zdobywający w ostatnich latach ogromną popularność Godot. Warto porównać wyniki uzyskane w Unity z rezultatami osiąganymi w Godocie ponieważ na moment obecny brak jest jakichkolwiek badań w tym temacie.

Bibliografia

[1] Newzoo, Global Games Market Report 2024, https://newzoo.com/resources/trend-reports/newzoos-global-games-market-report-2024-free-version, [16.06.2025].

[2] K. Claypool, M. Claypool, Perspectives, frame rates and resolutions: it's all in the game, In International Conference on Interactive Entertainment (FDG) (2009) 42–49, https://doi.org/10.1145/1536513.1536530.

[3] K. Claypool, M. Claypool, On frame rate and player performance in first person shooter games, Multimedia Systems 13 (2007) 3–17, https://doi.org/10.1007/s00530-007-0081-1.

[4] A. Madhusudan, B. Watson, Better frame rates or better visuals? An early report of esports player practice in Dota 2, In Extended Abstracts of the 2021 Annual Symposium on Computer-Human Interaction in Play (CHI PLAY) (2021) 174–178, https://doi.org/10.1145/3450337.3483484.

[5] K. Werder, Esport, Business & Information Systems Engineering 64(3) (2022) 393–399, https://doi.org/10.1007/s12599-022-00748-w.

[6] A. Trisnadoli, J. A. Kreshna, Optimization of Educational Mobile Game Design 'Ayo Wisata ke Riau' Based on User's Perspective, IT Journal Research and Development 6(1) (2021) 52–59, https://doi.org/10.25299/itjrd.2021.5766.

[7] J. Jie, K. Yang, H. Shi, Research on the 3D game scene optimization of mobile phone based on the Unity 3D engine, In International Conference on Computational and Information Sciences (ICCIS) IEEE (2011) 875–877, https://doi.org/10.1109/ICCIS.2011.317.

[8] D. Daman, T. K. Heok, A review on level of detail, In International Conference on Computer Graphics, Imaging and Visualization (CGIV) IEEE (2004) 14–20, https://doi.org/10.1109/CGIV.2004.1323963.