

Porównanie wydajności i możliwości współczesnych silników gier komputerowych

inż. Krzysztof Rudnicki

Promotor: dr inż. Michał Chwesiuk

Wydział Elektroniki i Technik Informatycznych
Politechnika Warszawska

Luty 2026

Plan prezentacji

- 1 Cel i zakres pracy
- 2 Metodologia
- 3 Implementacja
- 4 Narzędzie profilowania
- 5 Wyniki testów wydajności
- 6 Wywiady z deweloperami
- 7 Wnioski

Motywacja

- Rynek gier zdominowany przez dwa silniki:
 - **Unity** – prawie 25 000 gier na Steam
 - **Unreal Engine** – ponad 7 500 gier na Steam
- Brak systematycznych badań porównawczych
- Odmienne filozofie projektowe:
 - Unity: C#, garbage collector
 - Unreal: C++, dostęp do kodu źródłowego

Cel pracy

Porównanie wydajności i możliwości Unity oraz Unreal Engine:

- 1 Testy wydajności z NVIDIA Nsight Systems
- 2 Implementacja identycznej gry w obu silnikach
- 3 Analiza porównawcza funkcjonalności
- 4 Wywiady z 8 deweloperami gier

Hipoteza: Unity osiągnie lepszą wydajność

Wybór gatunku – bullet hell

Uzasadnienie:

- Setki/tysiące pocisków na ekranie
- Test zarządzania pamięcią (GC vs ręczne)
- Intensywne wykrywanie kolizji
- Skalowalność



Touhou Project – przykład bullet hell

Parametry gry testowej

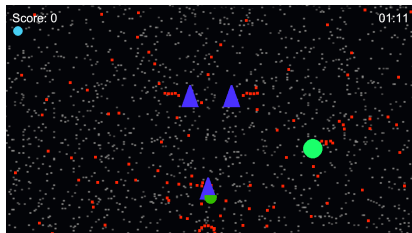
- Czas rozgrywki: **90 sekund**
- 3 typy przeciwników z różnymi wzorcami strzelania
- Eskalacja trudności w 3 fazach:
 - 1 **0–30s:** Niska trudność (podstawowi przeciwnicy)
 - 2 **30–60s:** Średnia (szybsze jednostki, wieżyczki)
 - 3 **60–90s:** Wysoka (wszystkie typy, max spawnu)

Środowisko testowe

Komponent	Specyfikacja
CPU	AMD Ryzen 9 7900X3D (24 rdzenie)
GPU	NVIDIA GeForce RTX 3090 (24 GB)
RAM	32 GB
OS	Arch Linux
Unity	6.0 LTS
Unreal	5.5.3
Profiler	NVIDIA Nsight Systems 2025.5.2

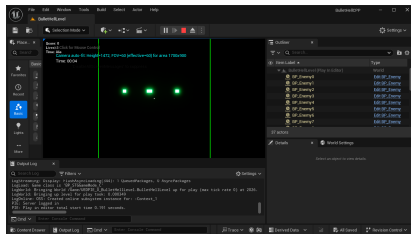
Implementacja – Unity

- Język: **C#**
- Natywny tryb 2D
- Rigidbody2D,
Collider2D
- Object pooling:
SetActive()
- Hot reload
- Instalacja: ok. 30 min



Implementacja – Unreal Engine

- Język: **C++** /
Blueprinty
- „Fałszywe 2D” w
środowisku 3D
- Object pooling – 3
metody:
 - `SetActorHiddenInGame`
 - `SetActorEnableCollision`



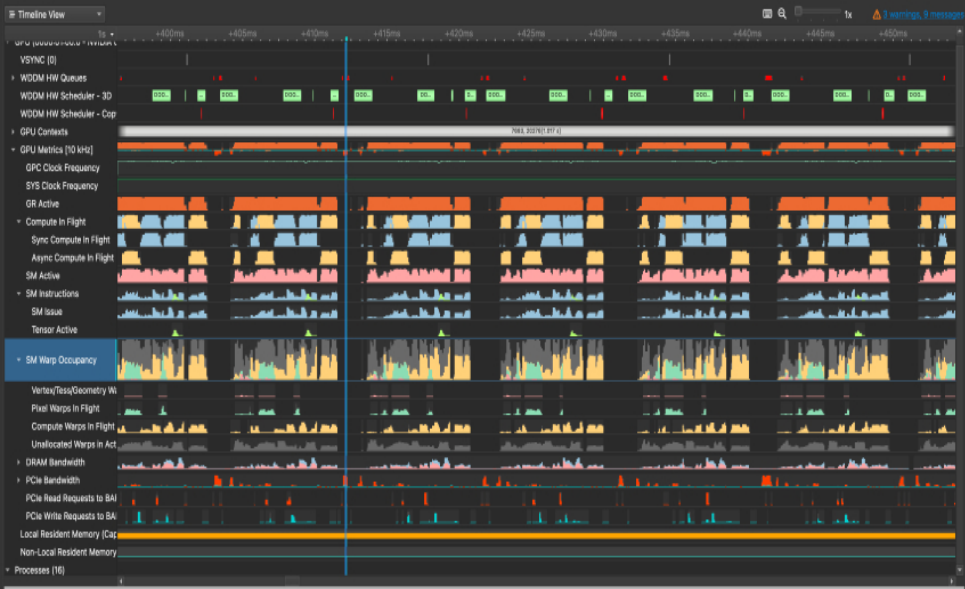
Porównanie doświadczeń implementacyjnych

Aspekt	Unity	Unreal
Instalacja (Linux)	~30 min	~2–4 h
Natywne 2D	Tak	Nie
Język	C#	C++
Próg wejścia	Niski	Średni/Wysoki
Czas kompilacji	Szybki	Wolny
Object pooling	Prosty	Złożony
Hot reload	Tak	Ograniczony

Implementacja w Unity zajęła ~**60%** czasu 10 / 24

Dlaczego NVIDIA Nsight Systems?

- Wbudowane profilery silników są **nieporównywalne**:
 - Różna definicja metryk
 - Różny narzut profilowania
 - Różne formaty wyjściowe
- NVIDIA Nsight – niezależne narzędzie:
 - Zunifikowane metryki sprzętowe
 - Analiza na poziomie Vulkan API
 - Minimalny narzut (poziom sterownika)
 - Spójny format dla obu silników



Expert System View

Settings

CUDA Async Memory with Pageable Memory

The following APIs use PAGEABLE memory which causes asynchronous CUDA memory operations to block and be executed synchronously. This leads to low GPU utilization.

Suggestion: If applicable, use PINNED memory instead.

CLI command:

SKIPPED: /Users/hnd/Downloads/Vulkantrace.sglite could not be analyzed because it does not contain the required CUDA data. Does the application use CUDA runtime?

12 / 24

Unity – wydajność klatek

Metryka	Wartość
Czas testu	94,16
Wyrenderowane klatki	13 550
Średni FPS	143,96 (V-Sync: 144 Hz)
Mediana czasu klatki	6,94 ms
99. percentyl (1% low)	7,58 ms (132 FPS)
Klatki w przedziale 5–10 ms	98,24%

Wysoka stabilność – IQR zaledwie 0,08 ms

Unity – architektura renderowania

- **Prosty potok:** 2 wywołania `vkQueueSubmit`/klatkę
- `vkWaitForFences` – **95,2%** czasu Vulkan API
 - CPU czeka na GPU → scenariusz **GPU-bound**
- Łącznie $\sim 218\,815$ wywołań Vulkan API
- Tylko **3 potoki graficzne** w całym teście
- Wykorzystanie GPU: $\sim 23\%$ (ograniczone_{14 / 24})

Unreal Engine – wydajność klatek

Metryka	Faza 1	Faza 2	Faza 3
Średni FPS	332	339	162
GPU Active	91%	91%	50%
vkQueueSubmit	166 918	186 589	74 393
Submit/klatkę	16,2	16,2	16,2

- Spadek o **ponad 50%** między fazami 1–2 a fazą 3
- Brak V-Sync – pełne wykorzystanie GPU

Unreal Engine – architektura renderowania

- **Złożony potok:** 16 wywołań `vkQueueSubmit`/klatkę
- Tworzenie potoków: **47–72%** czasu Vulkan API
 - ~1000 potoków na fazę (vs 3 w Unity!)
- Łącznie ~**32 mln** wywołań Vulkan API
- ~**9 mln** wywołań synchronizacji OS
- Intensywne użycie compute shaderów

Porównanie kluczowych wyników

Metryka	Unity	Unreal
FPS (niskie obciążenie)	164 (V-Sync)	332–339
FPS (wymagająca scena)	132 (1% low)	162 (faza 3)
Wykorzystanie GPU	23%	91% / 50%
Wywołania Vulkan API	~0,5 mln	~32 mln
Wywołania sync. OS	29 383	~9 mln
Potoki graficzne	3	~2 400
Submit/klatkę	2	10

Interpretacja wyników

Unity

- Prosty, dwuetapowy potok – wydajny dla gier 2D
- Stabilne czasy klatek (98,24% w 5–10 ms)
- Niewykorzystany potencjał GPU (V-Sync: 23%)

Unreal Engine

- Złożony, wieloetapowy potok – narzut dla prostych scen

Wywiady – najważniejsze wnioski

8 respondentów (1–10 lat doświadczenia):

- **Próg wejścia:** Unity niższy, Unreal wyższy
- **Dokumentacja:** Unity lepsza (przykłady kodu); Unreal – „szkieletowa”
- **Blueprinty:** Ułatwiają współpracę z nietechnicznymi, ale problemy z Git
- **Architektura:** Unreal wymusza porządek; Unity – elastyczny
- **C# vs C++:** C# łatwiejszy; C++ w Unreal „niestandardowy”

Weryfikacja hipotezy

„Unity osiągnie lepszą wydajność w grze bullet hell ...”

Hipoteza **częściowo potwierdzona**:

- ✓ Prostsza architektura renderowania
- ✓ Stabilniejsze czasy klatek
- ✓ Łatwiejszy proces implementacji (60% czasu)
- ✓ Natywne wsparcie 2D
- × Zbliżona wydajność w wymagających scenach (132 vs 162 FPS)
- × Unity ograniczone przez V-Sync

Rekomendacje

Typ projektu	Silnik	Uzasadnienie
Gra 2D indie	Unity	Natywne wsparcie 2D
Gra mobilna	Unity	Optymalizacja, rozmiar
Prototyp	Unity	Szybki cykl iteracji
Gra 3D AAA	Unreal	Nanite, Lumen
Gra VR high-end	Unreal	Zaawansowane oświetlenie
Zespół mieszany	Unreal	Blueprinty

Wkład pracy

- 1 **Zunifikowana metodyka pomiaru** –
Nsight Systems jako niezależne narzędzie
- 2 **Analiza Vulkan API** –
32 mln wywołań (Unreal) vs 0,5 mln
(Unity)
- 3 **Triangulacja metod** –
testy + wywiady + implementacja
- 4 **Praktyczne rekomendacje** –
macierz wyboru silnika

Ograniczenia i dalsze badania

Ograniczenia:

- Jeden gatunek gry (bullet hell)
- Pojedyncza konfiguracja sprzętowa (high-end)
- 8 wywiadów (badanie eksploracyjne)

Propozycje dalszych badań:

- Testy dla RPG, RTS, puzzle
- Różne platformy (mobile, konsole, low-end PC)
- Automatyczny framework benchmarkowy
- Badanie longitudinalne (2–3 lata)

Dziękuję za uwagę

Pytania?