Porównanie wydajności i możliwości współczesnych silników gier komputerowych

inż. Krzysztof Rudnicki

Promotor: dr inż. Michał Chwesiuk

Wydział Elektroniki i Technik Informacyjnych
Politechnika Warszawska

10 marca 2026

Plan prezentacji

Motywacja

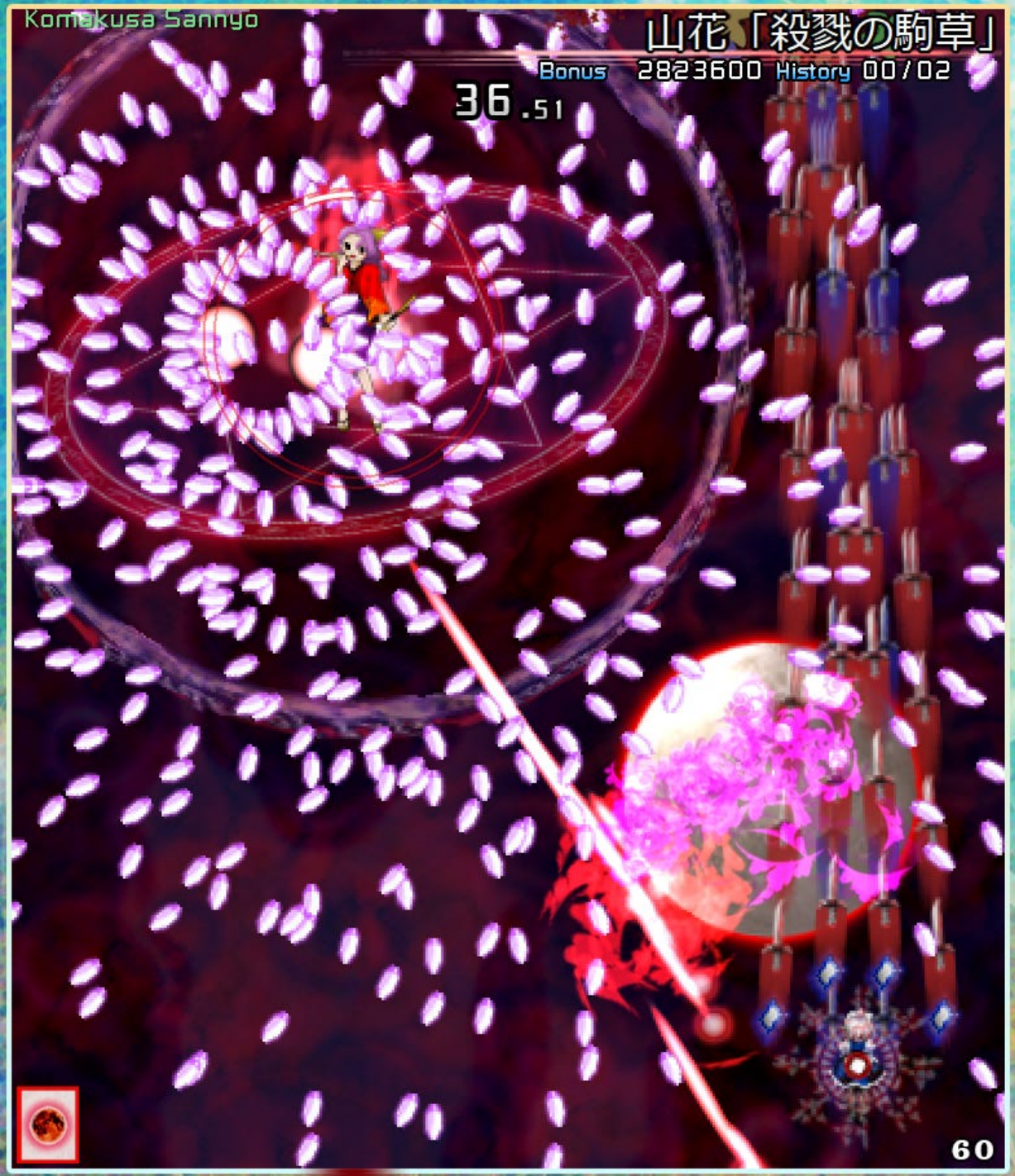
- Rynek gier zdominowany przez dwa silniki:
  - **Unity** – prawie 25 000 gier na Steam
  - **Unreal Engine** – ponad 7 500 gier na Steam
- Brak systematycznych badań porównawczych

Cel pracy

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 osiagnie lepsza wydajność w grze 2D *bullet hell* dzieki natywnemu wsparciu dla grafiki 2D.



NORMAL

ハイスコア 250,658,660

スコア 203,700,080

残り人数

(かけら) 2/3

スペルカード

(かけら) 1/3

P 霊力 4.00 / 4.00

資金力 661

使用カード

装備カード

能力カード

60.1fps

Parametry gry testowej

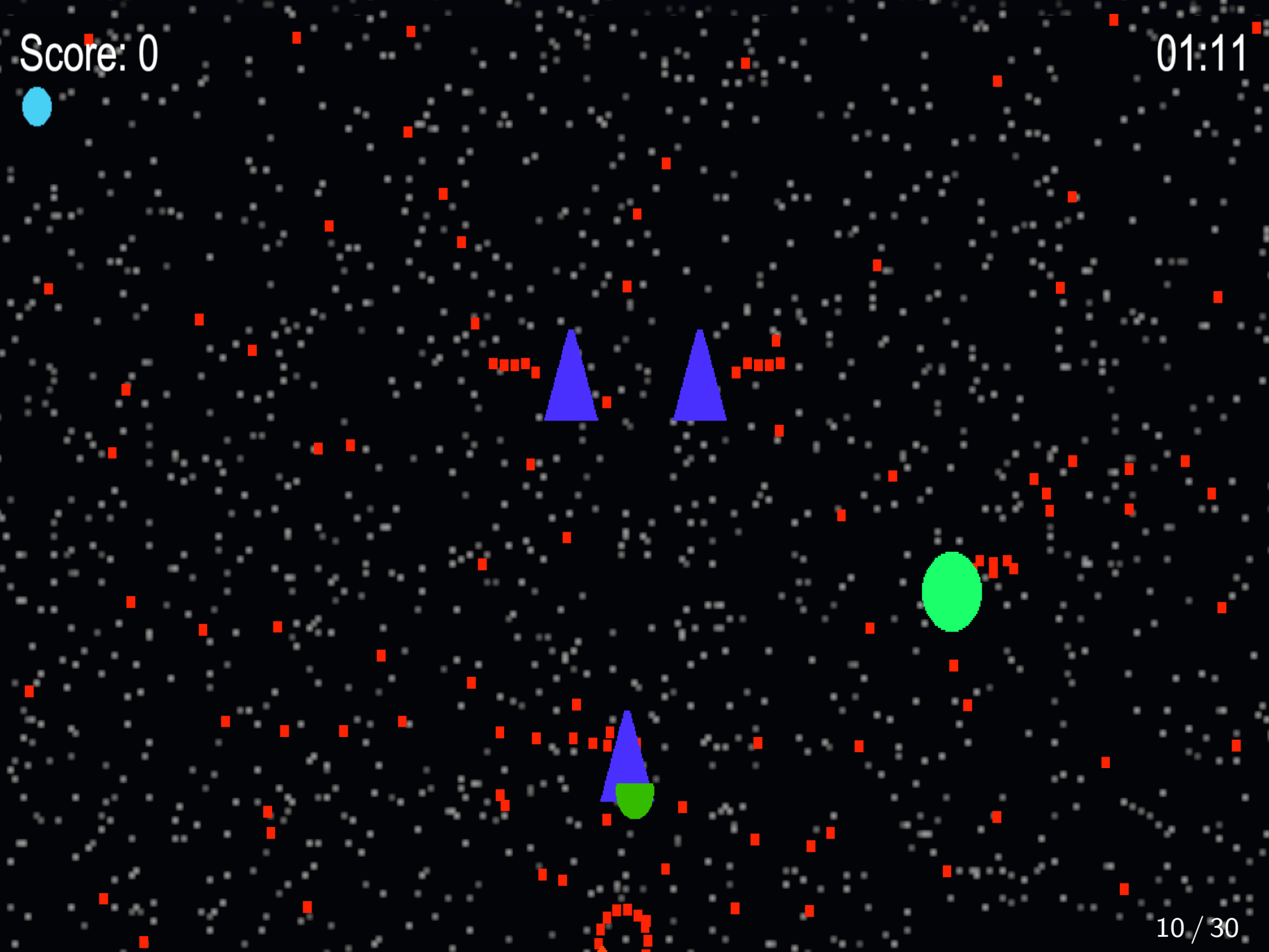
- Czas rozgrywki: **90 sekund**
- 3 typy przeciwników z różnymi wzorcami strzelania
- Eskalacja trudności w 3 fazach
- Limit: 200 jednoczesnych przeciwników
- Interwał spawnu: 0,25s $\rightarrow$ 0,08s

Środowisko testowe

CPU	AMD Ryzen 9 7900X3D
GPU	NVIDIA 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

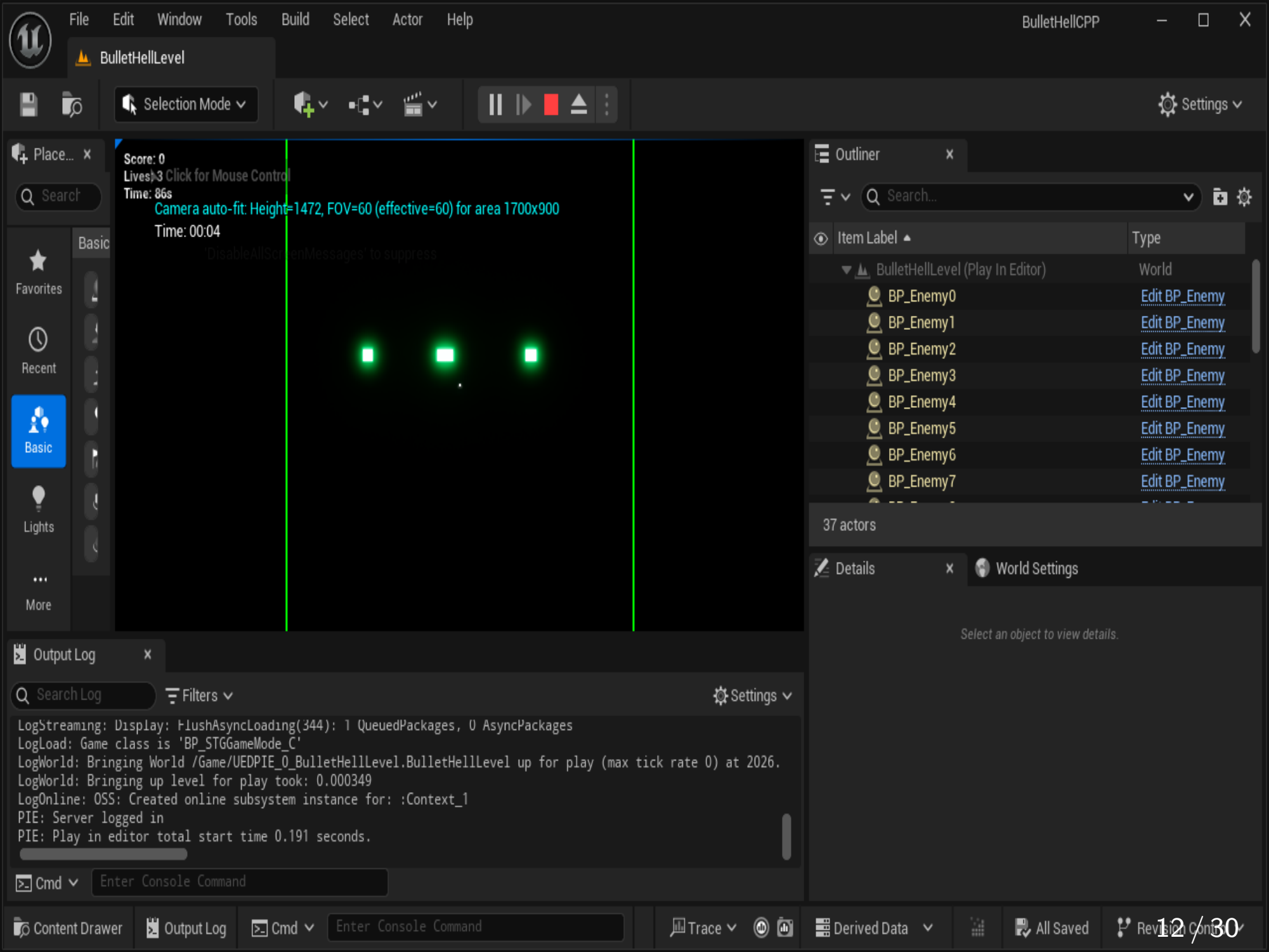
Implementacja – Unity

- Jezyk: **C#**
- Natywny tryb 2D
- `Rigidbody2D`, `Collider2D`
- Object pooling: `SetActive()`
- Hot reload
- Instalacja: $\sim$30 min


Score: 0
01:11
10 / 30

Implementacja – Unreal Engine

- Jezyk: **C++** / Blueprinty
- „Fałszywe 2D” w środowisku 3D
- Object pooling – 3 metody:
  - `SetActorHiddenInGame`
  - `SetActorEnableCollision`
  - `SetActorTickEnabled`
- Instalacja: $\sim$2–4h (Linux)

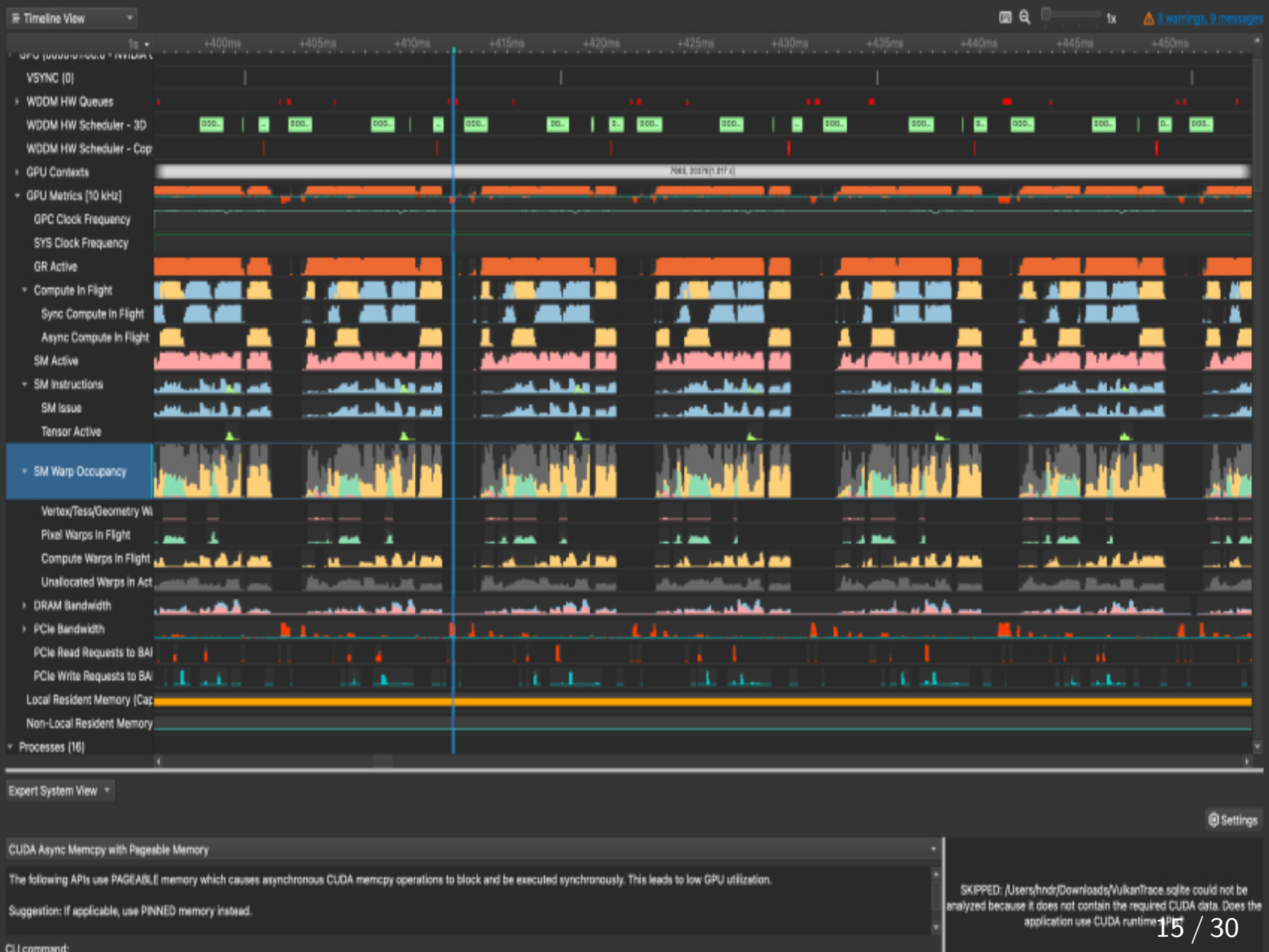

File Edit Window Tools Build Select Actor Help
BulletHellCPP
BulletHellLevel
Selection Mode
Settings
Place...
Search
Basic
Favorites
Recent
Basic
Lights
More
Score: 0
Lives: 3
Time: 86s
Click for Mouse Control
Camera auto-fit: Height=1472, FOV=60 (effective=60) for area 1700x900
Time: 00:04
DisableAllScreenMessages' to suppress
Outliner
Search...
Item Label
Type
BulletHellLevel (Play In Editor)
World
BP_Enemy0
BP_Enemy1
BP_Enemy2
BP_Enemy3
BP_Enemy4
BP_Enemy5
BP_Enemy6
BP_Enemy7
Edit BP_Enemy
37 actors
Details
World Settings
Select an object to view details.
Output Log
Search Log
Filters
Settings
LogStreaming: Display: FlushAsyncLoading(344): 1 QueuedPackages, 0 AsyncPackages
LogLoad: Game class is 'BP_STGGameMode_C'
LogWorld: Bringing World /Game/UEDPIE_0_BulletHellLevel.BulletHellLevel up for play (max tick rate 0) at 2026.
LogWorld: Bringing up level for play took: 0.000349
LogOnline: OSS: Created online subsystem instance for: :Context_1
PIE: Server logged in
PIE: Play in editor total start time 0.191 seconds.
Cmd
Enter Console Command
Content Drawer
Output Log
Cmd
Enter Console Command
Trace
Derived Data
All Saved
Revision Control

Porównanie doświadczeń implementacyjnych

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

Dlaczego NVIDIA Nsight Systems?

- NVIDIA Nsight – niezależne narzedzie:
  - Zunifikowane metryki sprzetowe
  - Analiza na poziomie Vulkan API
  - Minimalny narzut (poziom sterownika)
  - Spójny format dla obu silników


Timeline View
1x
3 warnings, 9 messages
+400ms
+405ms
+410ms
+415ms
+420ms
+425ms
+430ms
+435ms
+440ms
+445ms
+450ms
VSYNC (0)
WDDM HW Queues
WDDM HW Scheduler - 3D
WDDM HW Scheduler - Cop
GPU Contexts
GPU Metrics [10 kHz]
GPC Clock Frequency
SYS Clock Frequency
GR Active
Compute In Flight
Sync Compute In Flight
Async Compute In Flight
SM Active
SM Instructions
SM Issue
Tensor Active
SM Warp Occupancy
Vertex/Tess/Geometry Wa
Pixel Warps In Flight
Compute Warps In Flight
Unallocated Warps in Act
DRAM Bandwidth
PCIe Bandwidth
PCIe Read Requests to BA
PCIe Write Requests to BA
Local Resident Memory (Cap
Non-Local Resident Memory
Processes (16)
Expert System View
Settings
CUDA Async Memcpy with Pageable Memory
The following APIs use PAGEABLE memory which causes asynchronous CUDA memcpy operations to block and be executed synchronously. This leads to low GPU utilization.
Suggestion: If applicable, use PINNED memory instead.
SKIPPED: /Users/hndr/Downloads/VulkanTrace.sqlite could not be analyzed because it does not contain the required CUDA data. Does the application use CUDA runtime APIs?

Unity – wydajność klatek

Czas testu	94,16 s
Wyrenderowane klatki	13 556
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 5–10 ms	**98,24%**
IQR czasu klatki	0,08 ms

Unity – architektura renderowania

- **Prosty potok**: 2 wywołania `vkQueueSubmit`/klatke
- `vkWaitForFences` – **95,2%** czasu Vulkan API
  - CPU czeka na GPU → scenariusz **GPU-bound**
- Łacznie ∼218 815 wywołań Vulkan API
- Tylko **3 potoki graficzne** w całym teście
- Wykorzystanie GPU: ∼**23%** (ograniczone V-Sync)

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/klatke	16,2	16,2	16,2

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

Unreal Engine – architektura renderowania

- **Złożony potok**: 16 wywołań `vkQueueSubmit`/klatke
- Tworzenie potoków: **47–72%** czasu Vulkan API
  - ∼1000 potoków na faze (vs 3 w Unity!)
- Łacznie ∼**32 mln** wywołań Vulkan API
- ∼**9 mln** wywołań synchronizacji OS
- Intensywne użycie compute shaderów (culling, post-proc.)

Porównanie kluczowych wyników

Metryka	**Unity**	**Unreal**
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/klatke	2	16

Interpretacja – Unity

Unity

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

Interpretacja – Unreal Engine

Unreal Engine

- Złożony, wieloetapowy potok – narzut dla prostych scen
- Ciagła rekompilacja potoków (dynamiczna optymalizacja)
- Pełne wykorzystanie GPU ($\sim$91%)
- Spadek wydajności przy dużym obciażeniu ($>$50%)

Wywiady – najważniejsze wnioski (1/2)

8 respondentów (1–10 lat doświadczenia):

- **Próg wejścia**: Unity niższy, Unreal wyższy
- **Dokumentacja**: Unity lepsza (przykłady kodu); Unreal – ,,szkieletowa”
- **Blueprinty**: Ułatwiaja współprace z nietechnicznymi, ale problemy z Git

Wywiady – najważniejsze wnioski (2/2)

- **Architektura**: Unreal wymusza porzadek; Unity – elastyczny
- **C# vs C++**: C# łatwiejszy; C++ w Unreal ,,niestandardowy”
- **Materiały**: Unity przewaga ilościowa

Weryfikacja hipotezy

„Unity osiagnie lepsza wydajność w grze bullet hell . . .”

- Prostsza architektura renderowania
- Stabilniejsze czasy klatek
- Łatwiejszy proces implementacji (60% czasu)
- Natywne wsparcie 2D

Rekomendacje

Projekt	**Silnik**	**Powód**
2D indie	Unity	Natywne 2D
Mobilna	Unity	Optymalizacja
Prototyp	Unity	Szybkie iteracje
3D AAA	Unreal	Nanite, Lumen
VR high-end	Unreal	Oświetlenie
Zespół mieszany	Unreal	Blueprinty

Wkład pracy

1. **Zunifikowana metodyka pomiaru** – Nsight Systems jako niezależne narzedzie
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

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

Dalsze badania

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

Dziekuje za uwage

Pytania?