Jurnal SANTI (Sistem Informasi dan Teknologi Informasi)
Vol. 5 No.2 Tahun 2026
DOI: https://doi.org/10.58794/santi.v5i2.2190

Implementasi Niagara Particle System dalam Pengembangan Mini Game Aim Practice pada Unreal Engine 5

Raeldy Eliano Fazren [1], Mutaqin Akbar [2]
[1,2]Program Studi Informatika, Fakultas Teknologi Informasi, Universitas Mercu Buana Yogyakarta
e-mail: [1]221110020@mercubuana-yogya.ac.id, [2]mutaqin@mercubuana-yogya.ac.id

(Received :22 Mei 2026; Revised: 27 Mei 2026; Accepted: 29 Mei 2026; Available online: 26 Juni 2026)

Abstrak

Penelitian ini mengimplementasikan Niagara Particle System dalam pengembangan mini game aim practice berbasis First Person Shooter (FPS) berjudul Aim Practice pada Unreal Engine 5. Tiga efek visual utama dikembangkan, yaitu NS_HitMuzzle (kilatan senjata), NS_HitObject (tumbukan pada target), dan NS_HitGround (tumbukan pada permukaan), menggunakan C++ class AAimPracticeProjectile dan sistem event-driven UGameEventSystem berbasis Gameplay Tag. Metode pengembangan yang digunakan adalah Game Development Life Cycle (GDLC) yang mencakup empat tahap: Inisiasi, Pra-produksi, Produksi, dan Pengujian Alpha. Pengujian Black Box Testing terhadap 16 skenario menunjukkan seluruh fitur berfungsi dengan status Valid. Pengujian performa menggunakan Niagara Debugger mode Performance menunjukkan ketiga efek beroperasi dengan beban komputasi sangat kecil, Game Thread rata-rata di bawah 0,1 ms dan GPU time mendekati 0 ms. Hasil penelitian ini dapat menjadi referensi praktis bagi pengembang game dalam mengimplementasikan sistem efek visual yang efisien pada game FPS berbasis Unreal Engine 5.

Kata kunci: *Niagara Particle System, Unreal Engine 5, Game FPS, Aim Trainer, Efek Visual.*

Abstract

This study implements Niagara Particle System in the development of a First Person Shooter (FPS) mini game called Aim Practice on Unreal Engine 5. Three main visual effects were designed and implemented: NS_HitMuzzle (muzzle flash), NS_HitObject (impact on target), and NS_HitGround (impact on surface), utilizing C++ class AAimPracticeProjectile and an event-driven system UGameEventSystem based on Gameplay Tags. The Game Development Life Cycle (GDLC) method was applied across four stages: Initiation, Pre-production, Production, and Alpha Testing. Black Box Testing on 16 test scenarios verified that all features operated as expected. Performance testing using Niagara Debugger in Performance mode demonstrated that all three effects operate with minimal computational overhead, with average Game Thread values below 0.1 ms and GPU time approaching 0 ms. These results offer practical guidance for game developers seeking to implement efficient visual effect systems in FPS games built on Unreal Engine 5.

Keywords: *Niagara Particle System, Unreal Engine 5, FPS Game, Aim Trainer, Visual Effect.*

1. Pendahuluan

Industri game merupakan salah satu sektor teknologi informasi yang berkembang paling pesat di seluruh dunia, dengan semakin banyaknya pengembang yang memanfaatkan berbagai platform dan *game engine* dalam proses pengembangannya [1], [2], [3], [4], [5]. Unreal Engine, yang dikembangkan oleh Epic Games, dikenal memiliki kemampuan *rendering* grafis tingkat tinggi serta mendukung pengembangan game lintas platform. Unreal Engine 5 (UE5) merupakan versi terbaru yang dirilis secara resmi pada April 2022 dan menghadirkan berbagai fitur canggih, di antaranya Nanite, Lumen, dan Niagara Particle System [6].

Niagara Particle System merupakan sistem *visual effects* (VFX) generasi terbaru yang menggantikan sistem Cascade pada versi Unreal Engine sebelumnya. Sistem ini dirancang untuk memberikan fleksibilitas, skalabilitas, dan kontrol yang lebih tinggi dalam pembuatan efek visual secara *real-time* [7]. Kemampuan Niagara dalam menangani simulasi skala besar telah dibuktikan dalam penelitian pada platform metaverse berbasis UE5, di mana sistem ini mampu mensimulasikan lebih dari 40.000 karakter secara bersamaan dengan *frame rate* yang stabil di atas 60 fps [8].

Genre *First Person Shooter* (FPS) merupakan salah satu genre game paling populer, terutama dalam ekosistem *esports* yang terus berkembang. Pada genre ini, kemampuan *aiming* merupakan keterampilan fundamental yang sangat berpengaruh terhadap performa pemain [9], [10].

Banyak pemain memanfaatkan aplikasi *aim trainer* sebagai media latihan terstruktur. Game FPS sangat bergantung pada umpan balik visual yang responsif — termasuk efek visual *muzzle flash* dan efek tumbukan peluru — untuk memberikan pengalaman bermain yang imersif [11], [12]. Ketersediaan umpan balik visual yang tepat waktu dan akurat menjadi faktor penting yang membedakan game FPS berkualitas tinggi dari yang biasa.

Meskipun Niagara Particle System telah terbukti menghasilkan efek visual berkualitas tinggi, penelitian yang secara spesifik mengkaji implementasinya dalam konteks game FPS bertipe *aim practice* pada Unreal Engine 5 masih sangat terbatas [13]. Penelitian yang ada umumnya berfokus pada pengembangan game FPS secara umum [12], [13] atau pengujian efek partikel pada platform lain seperti Unity [11], [14], tanpa mengukur dampak performa Niagara secara mendalam menggunakan metrik GT, RT, dan GPU *time*.

Penelitian ini mengisi *gap* tersebut dengan mengimplementasikan dan mengevaluasi tiga efek Niagara secara spesifik dalam konteks *gameplay* FPS *aim practice*, disertai pengukuran performa menggunakan Niagara Debugger. Fokus penelitian diarahkan pada tiga efek visual utama: NS_HitMuzzle, NS_HitObject, dan NS_HitGround, yang diimplementasikan menggunakan kombinasi *Blueprint Visual Scripting* dan C++ dengan arsitektur *event-driven* berbasis *UGameEventSystem*.

2. Metode Penelitian

Penelitian ini menggunakan metode *Game Development Life Cycle* (GDLC) yang diusulkan oleh Ramadan dan Widyani [15], dan telah diterapkan dalam berbagai penelitian pengembangan game [16], [17], mencakup enam tahapan. Penelitian ini difokuskan pada empat tahap pertama yaitu inisiasi, pra-produksi, produksi, dan pengujian alpha, karena bertujuan mengevaluasi implementasi teknis Niagara Particle System melalui pengujian internal menggunakan metode *Black Box Testing* [18].

2.1. Tahap Inisiasi

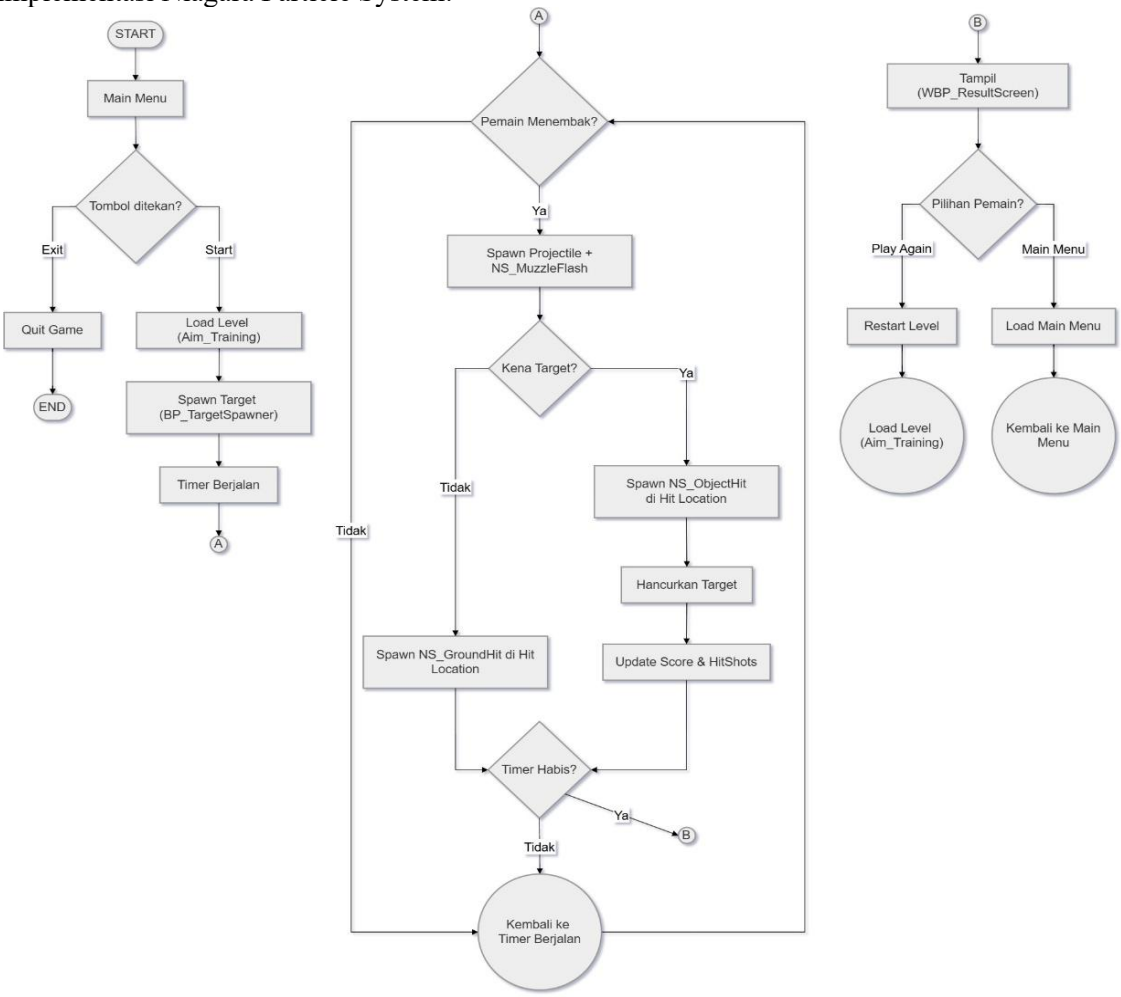
Pada tahap ini ditetapkan konsep, platform, *tools*, dan spesifikasi perangkat. Game yang dikembangkan adalah mini game *aim trainer* FPS bernama Aim Practice, di mana pemain bertugas menembak target yang muncul secara acak dalam batas waktu tertentu [9]. Platform target adalah PC Windows 11 dengan Unreal Engine 5.7, kombinasi *Blueprint Visual Scripting* dan C++. Spesifikasi perangkat pengembangan disajikan pada Tabel 1.

Tabel 1. Spesifikasi Perangkat Pengembangan

Komponen	Spesifikasi
Processor	AMD Ryzen 9 7950X 16-Core (32 CPUs)
Memory (RAM)	98304 MB (96 GB)
GPU	NVIDIA GeForce RTX 5070 Ti (VRAM 16 GB)
Operating System	Windows 11 Pro 64-bit (Build 26200)
DirectX	DirectX 12

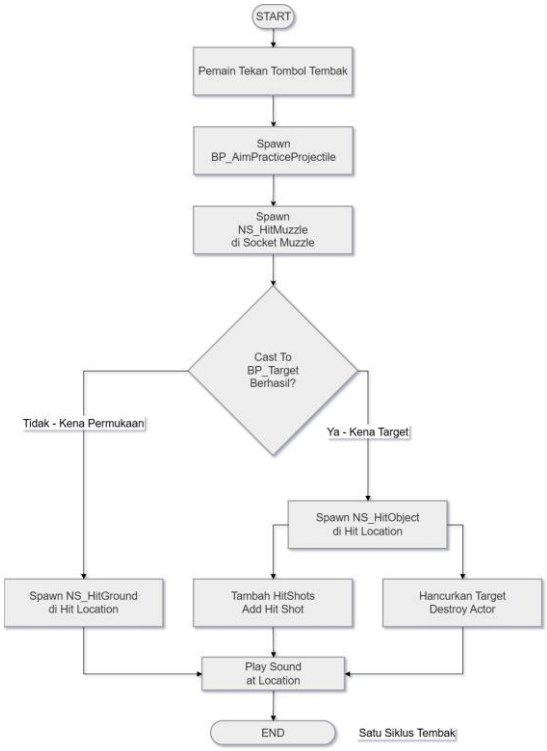
2.2. Tahap Pra-produksi

Pada tahap ini dirancang alur game, struktur komponen, dan rancangan efek Niagara. Alur game Aim Practice mencakup tiga bagian utama: *main menu*, *gameplay*, dan *result screen*. Gambar 1 menyajikan *flowchart* alur utama game, sedangkan Gambar 2 menyajikan alur implementasi Niagara Particle System.



Gambar 1. *Flowchart* Alur Utama Game

Gambar 1 menampilkan flowchart alur utama game Aim Practice yang terdiri dari tiga tahap utama. Pertama, pemain memulai dari Main Menu dan memilih tombol Start untuk masuk ke sesi gameplay. Selama gameplay berlangsung, timer berjalan mundur dan pemain menembak target yang muncul secara acak. Ketika timer habis, sistem secara otomatis menampilkan Result Screen yang memuat statistik permainan.



Gambar 2. *Flowchart* Alur Implementasi Niagara Particle System

Gambar 2 menampilkan flowchart alur implementasi Niagara Particle System yang menggambarkan bagaimana ketiga efek visual dipicu selama gameplay. NS_HitMuzzle di-spawn setiap kali pemain menekan tombol tembak, sedangkan NS_HitObject dan NS_HitGround dipicu berdasarkan hasil deteksi tumbukan peluru. Jika peluru mengenai objek bertag Target maka NS_HitObject yang di-spawn, jika tidak maka NS_HitGround yang di-spawn.

Struktur komponen game dikembangkan menggunakan kombinasi *Blueprint* dan C++ class. *Blueprint* menangani logika game umum seperti *scoring*, UI, dan alur permainan, sedangkan C++ menangani deteksi tumbukan *projectile* dan sistem komunikasi *event-driven* antar komponen [19]. Struktur komponen utama disajikan pada Tabel 2.

Tabel 2. Struktur Komponen Game Aim Practice

Komponen	Tipe	Fungsi
BP_FirstPersonCharacter	Blueprint	Input tembak, spawn projectile, spawn NS_HitMuzzle
AAimPracticeProjectile	C++ Class	Deteksi tumbukan, spawn NS_HitObject dan NS_HitGround
BP_AimPracticeProjectile	Blueprint (turunan C++)	Konfigurasi aset Niagara dan sound
BP_Target	Blueprint	Deteksi tumbukan, broadcast GameEventSystem, destroy target
BP_TargetSpawner	Blueprint	Spawn target berkala di area acak
BP_GameMode	Blueprint	Logika utama game, timer, scoring, perpindahan level
UGameEventSystem	C++ (WorldSubsystem)	Komunikasi event-driven berbasis Gameplay Tag
WBP_HUD	Widget Blueprint	Menampilkan Score dan Timer selama gameplay
WBP_ResultScreen	Widget Blueprint	Menampilkan Total Shots, Hit Shots, Accuracy, Score, High Score

Komponen	Tipe	Fungsi
WBP_MainMenu	Widget Blueprint	Tampilan dan logika tombol main menu

2.3. Tahap Produksi

Implementasi dilakukan menggunakan Unreal Engine 5.7 dengan dua pendekatan utama. Pertama, C++ class *AAimPracticeProjectile* dikembangkan untuk menangani logika deteksi tumbukan *projectile* dan *spawn* efek Niagara. Kelas ini diturunkan dari AActor dan memanfaatkan event *OnComponentHit* untuk memeriksa tag "Target" pada aktor yang tertumbuk; jika tag ditemukan maka NS_HitObject di-*spawn*, jika tidak maka NS_HitGround di-*spawn*, keduanya menggunakan *UNiagaraFunctionLibrary::SpawnSystemAtLocation*. Efek NS_HitMuzzle di-*spawn* langsung dari BP_FirstPersonCharacter pada *socket* Muzzle menggunakan node *Spawn System Attached* setiap kali input tembak diterima. Kedua, *UGameEventSystem* dikembangkan sebagai C++ *WorldSubsystem* yang menerapkan arsitektur *event-driven* menggunakan *Dynamic Multicast Delegate* dengan parameter *FGameplayTag*. Ketika target tertembak, BP_Target melakukan *broadcast* event dengan tag "Event.Target.Hitted", dan BP_GameMode yang telah *bind* ke *delegate* tersebut secara otomatis menerima notifikasi dan memperbarui *scoring*. Arsitektur ini memisahkan tanggung jawab deteksi tumbukan, komunikasi event, dan *scoring* ke komponen yang berbeda sesuai prinsip *separation of concerns* [18].

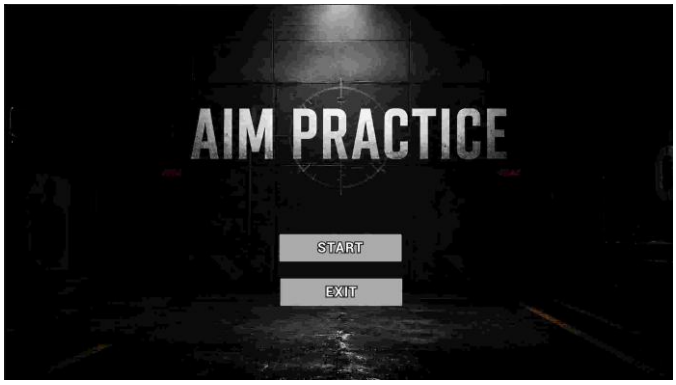
2.4. Tahap Pengujian Alpha

Pengujian dilakukan menggunakan dua metode. Metode pertama yakni pengujian *black box* untuk mengevaluasi fungsionalitas seluruh fitur berdasarkan input dan output yang dihasilkan tanpa memperhatikan struktur kode [18]. Skenario uji *black box* yang dilakukan terdiri dari 16 skenario uji, yaitu pengujian tombol Start, tombol Exit, efek NS_HitMuzzle saat menembak, efek NS_HitObject saat peluru mengenai target, efek NS_HitGround saat peluru mengenai permukaan, penambahan Score dan HitShots, penghancuran target, penambahan TotalShots, tampilan Result Screen saat timer habis, kalkulasi Accuracy, tombol Play Again, tombol Main Menu, spawn target di posisi acak, variasi ukuran target, persistensi High Score, serta kelancaran gameplay dengan ketiga efek Niagara aktif. Metode kedua yakni pengujian performa menggunakan Niagara Debugger mode Performance yang menampilkan metrik *Game Thread* (GT), *Render Thread* (RT), dan GPU *time* per sistem Niagara secara *real-time*. Ketiga parameter ini dipilih karena masing-masing merepresentasikan lapisan komputasi yang berbeda dalam *pipeline rendering* Unreal Engine 5. GT mengukur waktu pemrosesan logika partikel di sisi CPU, RT mengukur beban pengiriman perintah *render* ke GPU, sedangkan GPU *time* mengukur waktu eksekusi aktual di GPU. Kombinasi ketiganya memberikan gambaran menyeluruh mengenai distribusi beban komputasi efek Niagara, sehingga dapat diidentifikasi *bottleneck* yang mungkin terjadi pada setiap lapisan *pipeline*. Pengujian performa dilakukan dalam empat kondisi selama masing-masing satu menit dengan frekuensi tembakan yang konsisten.

3. Hasil dan Pembahasan

3.1. Hasil Implementasi

Hasil implementasi game Aim Practice mencakup pengembangan mini game FPS menggunakan Unreal Engine 5 dengan Niagara Particle System sebagai sistem efek visual utama. Game terdiri dari tiga tampilan utama—Main Menu, Gameplay, dan Result Screen—yang keseluruhannya diintegrasikan melalui arsitektur *Blueprint* dan C++ yang telah dirancang pada tahap Pra-produksi.



Gambar 3. Tampilan Main Menu

Gambar 3 menampilkan tampilan Main Menu dengan dua tombol utama: *Start* untuk memulai permainan dan *Exit* untuk keluar dari aplikasi. Animasi *fade in/out* diimplementasikan melalui WBP_MainMenu untuk memberikan transisi visual yang halus.

Gambar 4. Efek NS_HitMuzzle saat Pemain Menembak

Gambar 4 menampilkan efek NS_HitMuzzle berupa kilatan api yang muncul di ujung senjata setiap kali pemain melakukan tembakan. Efek ini dihasilkan dari sistem partikel yang di-spawn pada socket Muzzle senjata, memberikan umpan balik visual instan kepada pemain bahwa tembakan telah dilakukan.

Gambar 5. Efek NS_HitObject saat Peluru Mengenai Target

Gambar 5 menampilkan efek NS_HitObject yang muncul di lokasi tumbukan ketika peluru mengenai target. Efek ini mengandung beberapa emitter—percikan, asap, dan debris—yang secara kolektif memberikan konfirmasi visual bahwa tembakan tepat sasaran.

Gambar 6. Efek NS_HitGround saat Peluru Mengenai Permukaan

Gambar 6 menampilkan efek NS_HitGround yang muncul ketika peluru mengenai permukaan selain target. Efek ini berfungsi sebagai umpan balik visual negatif yang membantu pemain mengevaluasi akurasi bidikannya dan membedakan antara tembakan yang mengenai target dan yang tidak.

Gambar 7. Tampilan Result Screen

Gambar 7 menampilkan *Result Screen* yang muncul setelah timer habis, menampilkan Score, Accuracy, HitShots, TotalShots, dan High Score. Nilai Accuracy dihitung menggunakan rumus:

$$Accuracy\ \% = \frac{HitShots}{TotalShots} \times 100 \qquad (1)$$

High Score disimpan secara persisten menggunakan BP_SaveGame sehingga nilai tertinggi tetap tersedia meski game ditutup dan dibuka kembali.

3.2. Hasil Pengujian Black Box Testing

Tabel 3. Rancangan skenario Black Box Testing

No.	Skenario Uji	Yang Diharapkan	Status
1	Pemain menekan tombol Start	Level Aim_Training termuat, gameplay dimulai	Valid
2	Pemain menekan tombol Exit	Aplikasi game tertutup	Valid
3	Pemain menembak	NS_HitMuzzle muncul di ujung senjata	Valid
4	Peluru mengenai target	NS_HitObject muncul di lokasi tumbukan	Valid
5	Peluru mengenai permukaan selain target	NS_HitGround muncul di lokasi tumbukan	Valid
6	Target kena tembak	Score dan HitShots bertambah	Valid

No.	Skenario Uji	Yang Diharapkan	Status
7	Target kena tembak	Target hancur	Valid
8	Pemain menembak (mengenai atau tidak)	TotalShots bertambah	Valid
9	Timer mencapai 0	WBP_ResultScreen tampil	Valid
10	Kalkulasi Accuracy	Accuracy = (HitShots/TotalShots) × 100	Valid
11	Pemain menekan Play Again di result screen	Level Aim_Training dimuat ulang	Valid
12	Pemain menekan Main Menu di result screen	Main menu ditampilkan kembali	Valid
13	Target spawn	Target muncul di posisi acak dalam area spawn	Valid
14	Ukuran target	Setiap target memiliki ukuran berbeda-beda	Valid
15	High Score	High Score tersimpan dan tampil dengan benar	Valid
16	Gameplay dengan ketiga efek Niagara aktif	Game berjalan lancar tanpa lag atau crash	Valid

Berdasarkan hasil *Black Box Testing* terhadap 16 skenario uji (Tabel 3), seluruh fitur game Aim Practice menghasilkan *output* yang sesuai dengan yang diharapkan dan dinyatakan Valid. Tidak ditemukan kegagalan fungsionalitas pada seluruh fitur yang diuji, meliputi navigasi menu, ketiga efek Niagara, sistem *scoring*, persistensi High Score, dan alur game secara keseluruhan. Ketiga efek Niagara Particle System terbukti muncul tepat sesuai kondisi yang dipicu: NS_HitMuzzle muncul di ujung senjata setiap kali pemain menembak (skenario 3); NS_HitObject muncul di lokasi tumbukan ketika peluru mengenai target (skenario 4); dan NS_HitGround muncul ketika peluru mengenai permukaan selain target (skenario 5). Hasil ini membuktikan bahwa alur deteksi tumbukan melalui C++ class *AAimPracticeProjectile* berjalan dengan benar, demikian pula sistem komunikasi event melalui *UGameEventSystem* yang memastikan NS_HitObject tidak muncul bersamaan dengan NS_HitGround pada satu tembakan yang sama.

Selama proses pengembangan, terdapat beberapa tantangan teknis yang ditemui. Tantangan utama terletak pada perancangan arsitektur komunikasi antar komponen game, khususnya dalam menentukan mekanisme yang tepat untuk menghubungkan sistem deteksi tumbukan *projectile* dengan sistem *scoring*. Pendekatan awal menggunakan *Direct Cast* dari BP_Target ke BP_GameMode dinilai kurang fleksibel karena menciptakan ketergantungan langsung antar komponen. Solusi yang kemudian diterapkan adalah mengadopsi arsitektur *event-driven* berbasis *Dynamic Multicast Delegate* dengan parameter *Gameplay Tag* melalui UGameEventSystem, yang memungkinkan pemisahan tanggung jawab yang lebih bersih sesuai prinsip *separation of concerns*. Selain itu, proses penentuan *naming convention* untuk *delegate* juga melalui beberapa iterasi — dari OnTargetHitByCpp yang terlalu spesifik hingga akhirnya menggunakan OnTargetHitted yang lebih *generic* dan *extensible* untuk pengembangan *event* lanjutan.

3.3. Hasil Pengujian Performa

Pengujian performa dilakukan menggunakan Niagara Debugger mode Performance. Data disajikan sebagai nilai minimum, rata-rata, dan maksimum dari beberapa sesi pengujian selama masing-masing satu menit.

Gambar 8. Tampilan Niagara Debugger Mode Performance

Gambar 8 menampilkan tampilan Niagara Debugger mode Performance yang digunakan untuk mengukur performa ketiga sistem Niagara selama *gameplay*. Panel ini menampilkan metrik *Game Thread* (GT), *Render Thread* (RT), dan GPU *time* secara *real-time* untuk setiap sistem Niagara yang aktif. Data yang ditampilkan meliputi nilai minimum, rata-rata, dan maksimum dari setiap metrik sehingga memudahkan analisis performa efek visual.

Tabel 4. Hasil pengujian *Game Thread* (GT)

Sistem	Min (ms)	Avg (ms)	Max (ms)
Idle	0.000	0.000	0.000
NS_HitMuzzle	0.034	0.053	0.086
NS_HitObject	0.087	0.095	0.114
NS_HitGround	0.074	0.085	0.136

Tabel 4 menunjukkan hasil pengujian *Game Thread* (GT) untuk ketiga sistem Niagara. NS_HitObject mencatat beban GT tertinggi dengan nilai Avg 0,095 ms, diikuti NS_HitGround 0,085 ms, dan NS_HitMuzzle 0,053 ms. Urutan ini mencerminkan langsung kompleksitas *emitter* masing-masing efek: NS_HitObject dan NS_HitGround mengandung lebih banyak *emitter* (percikan partikel, asap, dan *debris*) dibandingkan NS_HitMuzzle yang hanya menghasilkan kilatan singkat.

Tabel 5. Hasil pengujian *Render Thread* (RT)

Sistem	Min (ms)	Avg (ms)	Max (ms)
Idle	0.000	0.000	0.000
NS_HitMuzzle	0.001	0.030	0.038
NS_HitObject	0.002	0.055	0.095
NS_HitGround	0.002	0.033	0.041

Tabel 5 menunjukkan hasil pengujian *Render Thread* (RT). NS_HitObject mencatat RT Avg 0,055 ms, hampir dua kali lipat NS_HitGround (0,033 ms) dan NS_HitMuzzle (0,030 ms). Nilai RT yang lebih tinggi pada NS_HitObject mengindikasikan beban *render* partikel yang lebih besar, kemungkinan karena jumlah partikel dan kompleksitas material *emitter* yang lebih tinggi dibandingkan kedua efek lainnya.

Tabel 6. Hasil pengujian GPU

Sistem	Min (ms)	Avg (ms)	Max (ms)
Idle	0.000	0.000	0.000
NS_HitMuzzle	0.000	0.010	0.012
NS_HitObject	0.000	0.004	0.006
NS_HitGround	0.000	0.002	0.009

Tabel 6 menunjukkan hasil pengujian GPU *time* ketiga sistem Niagara. Nilai GPU *time* pada ketiga efek sangat kecil dan mendekati 0 ms, tertinggi hanya 0,010 ms pada NS_HitMuzzle, menunjukkan bahwa komputasi partikel ditangani hampir seluruhnya oleh GT dan RT, bukan GPU.

Dari sudut pandang pengalaman *gameplay*, perbedaan beban komputasi antar ketiga sistem Niagara tidak berdampak terhadap kelancaran permainan. Seluruh nilai metrik berada jauh di bawah ambang batas yang dapat dirasakan pemain, bahkan nilai GT Max tertinggi (NS_HitGround 0,136 ms) hanya setara dengan kurang dari 1% dari anggaran waktu satu *frame* pada target 60 fps (16,67 ms). Perbedaan GT Avg antara NS_HitObject dan NS_HitMuzzle sebesar 0,042 ms (sekitar 79%) menunjukkan bahwa kompleksitas efek tumbukan berbanding lurus dengan beban komputasi *Game Thread*. Hasil ini sejalan dengan temuan Chung et al. yang membuktikan kemampuan Niagara dalam menangani simulasi berskala besar dengan performa stabil [8], serta penelitian Imam dan Areeb yang menunjukkan efek visual berbasis *game engine* mampu menghasilkan visualisasi imersif secara efisien [11], [14] dan Warburton et al. yang membuktikan umpan balik visual yang responsif berpengaruh terhadap performa *aiming* pemain dalam game FPS [20]. Perlu dicatat bahwa pengujian dilakukan pada perangkat dengan spesifikasi tinggi (RTX 5070 Ti, Ryzen 9 7950X), sehingga evaluasi lebih lanjut pada perangkat spesifikasi rendah diperlukan untuk mendapatkan gambaran performa yang lebih komprehensif.

4. Kesimpulan

Penelitian ini berhasil mengimplementasikan Niagara Particle System dalam pengembangan mini game Aim Practice pada Unreal Engine 5 melalui tiga efek visual utama — NS_HitMuzzle, NS_HitObject, dan NS_HitGround — menggunakan kombinasi C++ class AAimPracticeProjectile dan UGameEventSystem berbasis *Gameplay Tag*. Seluruh 16 skenario *Black Box Testing* dinyatakan Valid, membuktikan ketiga efek berfungsi responsif terhadap interaksi *gameplay*. Pengujian performa menunjukkan NS_HitObject mencatat beban tertinggi (GT Avg 0,095 ms, RT Avg 0,055 ms), sementara seluruh nilai GPU *time* mendekati 0 ms, membuktikan implementasi Niagara Particle System tidak memberikan beban signifikan pada performa game di bawah anggaran 16,67 ms per *frame* target 60 fps. Untuk penelitian selanjutnya, disarankan pengujian pada berbagai spesifikasi perangkat, eksplorasi *Scalability Settings* Niagara, serta pengembangan fitur *gameplay* tambahan seperti sistem *difficulty* dan mode *endless.*

Daftar Pustaka

[1] P. D. Novayanti and I. P. W. ADH, "Identifikasi Software dan Metode Pengembangan Game di Indonesia Menggunakan Systematic Literature Review," *JATISI (Jurnal Teknik Informatika dan Sistem Informasi)*, vol. 10, no. 4, pp. 301–320, 2023, doi: 10.35957/jatisi.v10i4.5957.

[2] A. Kartadinata and M. Akbar, "Implementasi Logika Finite State Machine dalam Perilaku Enemy Boss dalam Game 2D," *JITET*, vol. 13, no. 3, Jul. 2025, https://doi.org/10.23960/jitet.v13i3.7163.

[3] Z. Maulana and M. Akbar, "Implementasi Drop Rate Item menggunakan Algoritma Weighted Random Sampling dalam Game Action Platformer," *j.komputer, j.informasi, j.teknologi*, vol. 5, no. 1, p. 16, Jul. 2025, https://doi.org/10.53697/jkomitek.v5i1.2712.

[4] R. Adrian and M. Akbar, "Implementasi Sistem Character Player pada Game RPG 2D Menggunakan Game Engine Godot," *Jurnal Sains Informatika Terapan*, vol. 4, no. 3, pp. 800–805, 2025, doi: 10.62357/jsit.v4i3.663.

[5] K. Salen and E. Zimmerman, *Rules of Play: Game Design Fundamentals*. Cambridge, MA: MIT Press, 2004.
[6] O. Sobchyshak, S. Berrezueta-Guzman, and S. Wagner, "Pushing the boundaries of immersion and storytelling: A technical review of Unreal Engine," *Displays*, vol. 91, p. 103268, Jan. 2026, https://doi.org/10.1016/j.displa.2025.103268.
[7] X. Wei, "Interactive Volumetric Fog System in Unreal Engine 5 Using Distance Fields and GPU Particle Simulation," *Transactions on Computer Science and Intelligent Systems Research*, vol. 9, pp. 444–449, Jul. 2025, https://doi.org/10.62051/te0qnc44.
[8] K.-M. Chung, I.-K. Jeong, and J. Kim, "Implementation of a Massive Metaverse Platform using Niagara Particle Simulation in Unreal Engine," *Journal of Digital Contents Society*, vol. 25, no. 5, pp. 1387–1397, May 2024, https://doi.org/10.9728/dcs.2024.25.5.1387.
[9] E. J. Rogers, M. G. Trotter, D. Johnson, B. Desbrow, and N. King, "KovaaK's aim trainer as a reliable metrics platform for assessing shooting proficiency in esports players: a pilot study," *Front. Sports Act. Living*, vol. 6, p. 1309991, Feb. 2024, https://doi.org/10.3389/fspor.2024.1309991.
[10] H. K. Rhee, D. H. Song, and J. H. Kim, "Comparative Analysis of First Person Shooter Games on Game Modes and Weapons—Military-Themed, Overwatch, and Player Unknowns' Battleground," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 13, no. 1, pp. 116–122, 2019, https://doi.org/10.11591/ijeecs.v13.i1.pp116-122.
[11] R. Imam and Q. M. Areeb, "Game Effect Based on Particle System in Unity 3D," *Academia Letters*, Jul. 2021, https://doi.org/10.20935/AL2447.
[12] N. Raut, S. Chopade, T. Nichat, H. Kuhad, B. Shreeniwas, and D. Choudhari, "3D First-Person Shooter Game Development Using Unreal Engine 5," *IJRASET*, vol. 11, no. 5, pp. 4106–4115, May 2023, https://doi.org/10.22214/ijraset.2023.51039.
[13] A. D. Fernandez and J. Sutopo, "The Use of Visual Scripting Unreal Engine 5 in the Development of an Android-Based FPS Game Left Behind," *IJAR*, vol. 11, no. 11, pp. 1157–1165, Nov. 2023, https://doi.org/10.21474/IJAR01/17922.
[14] B. Zhang and W. Hu, "Game special effect simulation based on particle system of Unity3D," in *2017 IEEE/ACIS 16th International Conference on Computer and Information Science (ICIS)*, Wuhan, China: IEEE, May 2017, pp. 595–598. https://doi.org/10.1109/ICIS.2017.7960062
[15] R. Ramadan and Y. Widyani, "Game development life cycle guidelines," in *2013 International Conference on Advanced Computer Science and Information Systems (ICACSIS)*, Sanur Bali, Indonesia: IEEE, Sep. 2013, pp. 95–100. https://doi.org/10.1109/ICACSIS.2013.6761558.
[16] R. Y. Ariyana, E. Susanti, Muhammad Rizqy Ath-Thaariq, and R. Apriadi, "Penerapan Metode Game Devlopment Life Cycle (GDLC) pada Pengembangan Game Motif Batik Khas Yogyakarta," *INSOLOGI: Jurnal Sains dan Teknologi*, vol. 1, no. 6, pp. 796–807, Dec. 2022, https://doi.org/10.55123/insologi.v1i6.1129.
[17] S. Syarif, T. Hasanuddin, and M. Hasnawi, "Perancangan Game Puzzle Labirin menggunakan Metode Game Development Life Cycle (GDLC) berbasis Unreal Engine," *BUSITI*, vol. 3, no. 1, pp. 34–41, Feb. 2022, https://doi.org/10.33096/busiti.v3i1.582.
[18] R. S. Pressman, *Software engineering: a practitioner's approach*, 7th ed. New York: McGraw-Hill Higher Education, 2010.
[19] S. Butler, *Game Development Patterns with Unreal Engine 5: Build maintainable and scalable systems with C++ and Blueprint*, 1st ed. Birmingham: Packt Publishing Limited, 2024. https://doi.org/10.1007/978-1-4842-9824-4_1.
[20] M. Warburton, C. Campagnoli, M. Mon-Williams, F. Mushtaq, and J. R. Morehead, "Kinematic markers of skill in first-person shooter video games," *PNAS Nexus*, vol. 2, no. 8, p. pgad249, 2023, https://doi.org/10.1093/pnasnexus/pgad249.