© APR 2026 | IRE Journals | Volume 9 Issue 10 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I10-1716251

Design and Implementation of A 3D Shooter Game Using Unity Engine

YOGESH SHAH[1], AMIT YADAV[2], DEEPESH WADHE[3], SHAILESH CHAVAN[4] DR. SUNANDA PANDITA[5], BHAKTI DESHMUKH[6]

[1,2,3,4,5,6]*Department of Computer Science and Design EngineeringNew Horizon Institute of Technology and Management, Thane, India*

Abstract- ***The gaming industry has rapidly evolved with the advancement of graphics technologies and interactives gameplay systems. This paper presents the design and implementation of a 3D shooter game developed using the unity game engines. The system focuses on creating an immersive gameplay experience where players navigate through environments, interact with objects, and eliminate enemies using different weapons. The game incorporates artificial intelligence for enemy behaviour, real time score tracking, and player health management. The proposed system demonstrates how modern game engines can be utilized to developed interactive and scalable gaming applications with efficient performance and engaging user experiences.***

Index Terms- ***Shooter Game, Unity Engine, Game Development, Artificial Intelligence, Player Interaction.***

I. INTRODUCTION

Video game has become one of the most popular forms of entertainment worldwide. With the advancement of computer graphics and game engines, developers can now create highly immersive gaming experiences. Shooter games are among the most widely played game genres, where players interact with the game environment and defeat enemies using weapons.

A shooter game typically involves navigation, aiming, shooting mechanics, enemy artificial intelligence, and score management systems. These games require real time interaction between the player and the virtual environment. This project focuses on the design and development of a 3D shooter game using the Unity game engine.

Unity game provides a powerful platform for game development with built-in physics, animation systems, and scripting capabilities. The developed game includes features such as enemy Spawning, Weapon systems, health tracking, and mission

completion mechanisms. The main objectives of this project is to create an engaging shooter game that demonstrates fundamental concepts of game, design, artificial intelligence, and real time interaction.

Unlike conventional game architectures, the proposed 3D shooter system built using the Unity engine introduces several advanced features, including adaptive enemy AI behaviour, real-time physics-based interactions, dynamic difficulty scaling, and modular weapon systems. These enhancements enable the game to deliver not only engaging combat mechanics but also a responsive and immersive player experience. Additionally, features such as environment-aware navigation, hitbox precision, and animation blending contribute to realism and fluid gameplay.

The core objective of this project is to design a scalable and performance-optimized 3D shooter framework that ensures smooth gameplay across a wide range of hardware configurations while maintaining high visual fidelity and responsiveness. By integrating efficient rendering techniques, object pooling, event-driven scripting, and optimized asset management, the system aims to outperform traditional shooter implementations in both performance and user engagement.

The remainder of this project presents a detailed overview of existing 3D shooter design approaches, followed by the proposed system architecture, implementation methodology, testing procedures, and performance evaluation, demonstrating the effectiveness of the Unity-based framework in

delivering a robust and immersive gaming experience.

II. LITERATURE REVIEW / RELATED WORK

A. Game Design Frameworks and Traditional Shooter Architectures

Several frameworks have been developed for building 3D shooter games using engines like Unity. Traditional architectures often rely on component-based systems, where player controls, enemy behavior, and environment interactions are handled through separate scripts. Standard implementations include finite state machines (FSM) for enemy AI, basic collision detection, and predefined level design. While these approaches are effective for simple gameplay scenarios, they are limited in handling adaptive AI behavior, dynamic environments, and large-scale interactions. Similarly, platforms like Discourse implement activity-based trust levels (e.g., TL0–TL4) to promote community moderation. Although simple and scalable, these systems rely on static thresholds and lack probabilistic modelling of trust, making them less adaptive to dynamic user behavior.

Other heuristic-based approaches focus on improving gameplay through manual tuning of parameters such as enemy difficulty, weapon balance, and player progression. While practical, these methods are often time-consuming, subjective, and difficult to scale across complex game environments.

B. AI and Advanced Gameplay Systems in Shooter Games.

Recent advancements in game development have introduced AI-driven techniques to enhance player experience in 3D shooter games. Techniques such as behavior trees, navigation meshes (NavMesh), and reinforcement learning enable more realistic and adaptive enemy behavior. These systems allow enemies to react dynamically to player actions, improving immersion and challenge.

Procedural content generation is also increasingly used to create dynamic levels, missions, and enemy spawn patterns. While this enhances replayability, many implementations lack fine control over gameplay balance and may produce inconsistent player experiences.

Multiplayer systems and networking models (e.g., client-server architecture) have also been explored to enable real-time player interaction. Despite their advantages, issues such as latency, synchronization, and cheating prevention remain significant challenges in large-scale implementations.

C. Research Gaps

Despite significant progress in Unity-based shooter game development, several limitations remain:

1. Lack of adaptive and self-learning AI systems that evolve based on player behavior.
2. Over-reliance on static difficulty settings, limiting personalized gameplay experiences.
3. Absence of dynamic environment interaction and procedural adaptability in many systems.
4. Limited focus on performance optimization for scalability across diverse hardware platforms.

The proposed Unity-based 3D shooter system addresses these gaps through a modular and scalable architecture that integrates adaptive AI, dynamic difficulty adjustment, and optimized rendering techniques. By incorporating object pooling, event-driven scripting, NavMesh-based navigation, and real-time performance monitoring, the system ensures both high responsiveness and efficiency. Additionally, it introduces intelligent enemy behavior, flexible weapon systems, and immersive environment design, resulting in a robust and engaging gameplay experience.

Table I: Comparison of Existing Code Review Tools and the Proposed System

Game Engine	Approach	Limitation	Autonomy Level
Unreal Engine	High-end rendering engine	Requires high systems resources.	High
CryEngine	Advanced graphics engine	Complex development environment	High
Godot	Open-	Limited	Medium

© APR 2026 | IRE Journals | Volume 9 Issue 10 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I10-1716251

Engine	source Engine	built-in assets	
Proposed System (Unity Shooter Game)	Unity Engine with modular scripts	Requires asset optimization	High

III. PROPOSED SYSTEM

The proposed system, BattleStack, is designed as a credibility-aware competitive shooter framework that performs automated player skill evaluation and gameplay integrity assessment through an iterative, performance-driven mechanism. The system integrates player actions such as kills, assists, objective contributions, and team interactions into a unified pipeline that continuously refines both player skill ratings and match impact scores.

Unlike traditional shooter ranking systems, BattleStack does not rely on static leaderboards or simple metrics like kill/death ratio. Instead, it implements a self-correcting feedback mechanism, where in-game performance influences match impact scores, and the resulting outcomes dynamically update player skill ratings. This iterative process ensures robustness against exploitation, smurfing, and unbalanced playstyles.

The core architecture is inspired by iterative ranking models and extends them into a real-time gaming environment by incorporating weighted performance metrics, dynamic skill updates, impact score accumulation, and time-decay ranking. The system operates autonomously once gameplay data is recorded, requiring minimal manual intervention for player evaluation and matchmaking optimization.

A. System Components

1) Player Engine (Input and Skill Initialization)

The Player Engine represents all participants in the shooter game. Each player is assigned an initial skill rating $s_p \in [0,1]$and a fair-play score $f_p \in [0,1]$.

- Players interact through matches, contributing actions such as kills, assists, objectives, and teamwork.
- Initially, all players are treated equally, but over time:
  - Skill rating evolves based on performance
  - Fair-play score evolves based on behaviour (toxicity, cheating, quitting).

2) Performance Engine (Match Evaluation Module)

The Performance Engine evaluates player impact in each match using weighted gameplay metrics.
Instead of relying only on kills, it considers:

- Combat effectiveness (kills, damage, accuracy)
- Objective contribution (captures, defuses, payload time)
- Teamplay (assists, revives, support actions)

Each player's match performance is computed as:

- Weighted aggregation of gameplay actions
- Normalization relative to match context (team average, difficulty)

3) Iterative Skill & Fair-Play Update Module (Self-Correcting System)

This module updates both skill rating and fair-play score after each match.

- Players performing consistently well gain skill gradually
- Poor performance results in gradual skill decline
- Fair-play updates:
  - Positive behaviour (teamplay, no reports) → slow increase
  - Negative behaviour (toxicity, AFK, cheating flags) → rapid decre

4) CombatScore and Ranking Module (Engagement & Temporal Dynamics)

Each player accumulates a CombatScore, representing long-term contribution and activity.
CombatScore increases with:

- Consistent performance
- Objective play
- Positive team impact

Decreases with:

- Repeated poor performance
- Negative behaviour

Match and leaderboard rankings integrate:

© APR 2026 | IRE Journals | Volume 9 Issue 10 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I10-1716251

- Skill rating
- Recent performance (time-weighted)
- Match impact
- Fair-play score (penalizes toxic players)
- Activity level

5) Match Insight and Behavior Classification Module (Interpretation Layer)

This module translates raw metrics into understandable labels.
Match Classification:

- High Dominance – one team clearly outperformed
- Competitive – close and balanced match
- Unstable – inconsistent performance across players
- Upset – lower-ranked team wins

Player Classification:

- Elite Player – high skill, consistent performance
- Support Specialist – strong team contribution
- Aggressor – high combat focus
- Inconsistent – fluctuating performance

Behaviour Tags**:**

- High Fair Play – cooperative and clean gameplay
- Mixed Behaviour – occasional issues
- Disruptive – toxic or rule-breaking behaviour

This layer improves transparency and helps players understand both match quality and individual performance.

IV. METHODOLOGY AND SYSTEM ARCHITECTURE

A. System Architecture Description

The proposed shooter game system follows a modular, layered architecture designed for dynamic skill evaluation, fair-play assessment, and matchmaking in multiplayer environments. The Presentation Layer serves as the game client interface, displaying real-time outputs such as health, score, leaderboards, skill ratings, and fair-play indicators while capturing player actions. The Application Layer manages core functionalities including action handling, match state management, and session tracking, acting as the bridge between player input and analytics. The Processing Layer implements the iterative performance-aware skill and fair-play model, aggregating player contributions, updating skill ratings, and detecting disruptive behavior over multiple iterations until convergence.

System Workflow

The end-to-end data flow consists of four sequential stages:
Stage1(PlayerInteraction):
Players perform actions such as shooting, objective completion, assists, and support moves. These actions are logged along with initial skill ratings and fair-play scores.
Stage2(PerformanceComputation):
The system computes match performance for each player using weighted aggregation of contributions (combat, objectives, support), normalized by team skill and activity levels.
Stage3(Iterative Skill & Fair-Play Update):
Player skill and fair-play scores are updated iteratively based on alignment with match outcomes and contribution effectiveness. High-performing and fair players gain incremental rating increases, while low-performing or disruptive players are penalized.
Stage4(RankingandClassification):
Players are ranked and categorized using skill rating, CombatScore, fair-play score, recent performance (time decay), and contribution type. Matches are classified as balanced, competitive, dominant, or upset.
This pipeline ensures continuous refinement of skill evaluation and matchmaking, making the system adaptive, competitive, and resistant to disruptive behaviour.

B Algorithm: Shooter Game Performance Evaluation Workflow

Algorithm1:SkillandFair-PlayEvaluation Input:

- Set of players $\boldsymbol{P}$, player actions $\boldsymbol{a_{p,m}}$, initial skill ratings $\boldsymbol{s_p}$, initial fair-play scores $\boldsymbol{f_p}$

Output:

- Final player skill ratings $\boldsymbol{s'_p}$, updated fair-play scores $\boldsymbol{f'_p}$, match performance scores $\boldsymbol{M_m}$, and ranking scores $\boldsymbol{R_p}$

© APR 2026 | IRE Journals | Volume 9 Issue 10 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I10-1716251

Step1[Initialization]:
1.1 Assign initial skill rating $s_p \in [0,1]$and fair-play score$f_p \in [0,1]$forallplayers
1.2 Set initial performance score $M_m = 0.5$for each match
1.3 Set iteration counter $k = 0$

Step 2 [Weighted Credibility Computation]:
2.1 Compute weighted contributions:

- Offensive score O_p(kills, damage, accuracy)
- Objective score Obj_p(captures, defuses, time on objective)
- Support score Sup_p(assists, revives, utility usage)

Step3[AlignmentCheck]:
3.1 For each player p:

- If player performance aligns with match outcome M_m, mark as aligned
- Else mark as misaligned

Step 4 [Skill & Fair-Play Update]:
Update skill rating s_p:

- Reward aligned players with gradual increase
- Penalize misaligned players with gradual decrease

4.2 Update fair-play score f_p:

- Increase for positive behavior (teamplay, no reports)

Step 5 [CombatScore Update]:
5.1 Increase CombatScore for aligned players
5.2 Decrease CombatScore for misaligned players

Step6[ConvergenceCheck]:
6.1 Check if:

- Skill ratings stabilize across iterations

6.2 If not converged:

- Increment iteration $k = k + 1$

Step7[RankingandClassification]:
7.1 Compute ranking score $R_p$using skill rating

C. Development Methodology
The project follows an Agile Software Development Lifecycle (SDLC) with iterative refinement. This approach enables continuous validation of gameplay mechanics, player balancing, and anti-cheat systems.

Table II: Agile Development Methodology – Phase Activities (Shooter Game)

Phase	Key Activities
Requirement Analysis	Identify player expectations (gameplay, fairness, performance); define acceptance criteria for combat system, matchmaking, and anti-cheat modules
System Design	Architect core game systems (combat engine, matchmaking, scoring, networking); define data models for players, matches, and events
Implementation	Develop modules such as shooting mechanics, physics engine, player movement, and matchmaking using game engines (e.g., Unity/Unreal) and backend services
Integration	Integrate gameplay systems with backend servers; enable multiplayer networking; deploy using containers/servers with real-time match synchronization
Testing	Conduct playtesting and automated tests; validate weapon balance, hit detection, latency, and fairness; simulate player behaviour and stress test servers
Deployment & Feedback	Release to beta testers or live servers; collect player feedback, gameplay analytics, and satisfaction metrics for continuous improvement

D. Technology Stack (Shooter Game – Unreal Engine)
The system is implemented using the following technologies:

- Frontend: Game client built using Unreal Engine for graphics, UI, and player controls

© APR 2026 | IRE Journals | Volume 9 Issue 10 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I10-1716251

- Backend: Server-side logic using Spring Boot or Node.js for matchmaking and game management
- Database: Relational database (e.g., MySQL) for storing player and match data
- Networking: Real-time multiplayer using Unreal Engine's client-server architecture
- Processing: Algorithms for skill rating, scoring, and fair-play evaluation
- Visualization: Leaderboards and in-game performance statistics.

E. Implementation and Source Code

The shooter game system is implemented as a prototype platform integrating player interaction, performance evaluation, and skill progression modules. The system supports real-time gameplay processing, continuous updates, and classification of player and match outcomes.

The implementation includes:

- Player control and combat interface
- Performance evaluation engine (kills, objectives, teamwork)
- Skill and fair-play update mechanism
- Ranking and match classification module

The modular design ensures scalability and allows future integration with AI-based bots, advanced matchmaking algorithms, and player behaviour analytics.

The proposed architecture ensures modularity, scalability, and real-time stability, making it suitable for deployment in large-scale multiplayer shooter environments.

V. EVALUATION METHODOLOGY

The proposed shooter game system was evaluated using a simulated gameplay dataset representing player behaviour, including both normal and disruptive players (e.g., toxic, AFK, or exploitative users). The dataset includes multiple matches with controlled performance variations to test skill rating accuracy, matchmaking quality, and fair-play enforcement under noisy conditions

A. Evaluation Metrics

The system performance is measured using standard metrics:

- Accuracy (Skill Prediction): Correctness of player skill estimation based on match outcomes
- Match Balance Score: Degree of fairness between competing teams
- Convergence Stability: Stability of player skill ratings over iterations
- Fair-Play Robustness: Ability to detect and reduce impact of disruptive players

B. Experimental Setup

- Players: 10,000 (including 20% disruptive players)
- Matches: Simulated multiplayer sessions with varied skill distributions
- Iterations: $K_{max} = 5$
- Threshold: $\epsilon = 0.001$
- Parameters: Skill update rate $\alpha = 0.01$, fair-play penalty rate $\beta = 0.02$

C. Baseline Comparison

The system is compared with traditional ELO-based matchmaking systems.
This proposed model extends baseline approaches by incorporating:

- Performance-based multi-factor evaluation (not just win/loss)
- Fair-play scoring and behaviour tracking

.

D. Summary

The evaluation demonstrates that the shooter game system achieves accurate skill estimation, balanced matchmaking, stable convergence, and strong resistance to disruptive behaviour, making it suitable for large-scale multiplayer shooter deployments.

VI. RESULTS AND DISCUSSION

A. Quantitative Results

- Skill prediction AUC: 0.95 → effectively distinguishes high- vs low-skill players
- Disruptive behaviour detection recall: 0.91 → outperforms baseline ELO (≈0.84)
- Match balance F1-score: 0.88 → ensures fair and competitive matches

- Skill stability: ~0.93 → reliable player ranking over iterations
- CombatScore correlation with engagement: 0.80 → reflects long-term participation and contribution.

B. Qualitative Results

Qualitative analysis shows that the shooter game system effectively distinguishes between high-skill, consistent players, support specialists, aggressive players, and inconsistent performers.

The system successfully reduces the impact of disruptive players (toxic, AFK, or cheating) while preserving meaningful contributions from less prominent players, such as strategic support or objective-focused actions.

C. Discussion

The results highlight the effectiveness of the iterative skill- and fair-play–based framework in improving both accuracy and robustness of player ranking and matchmaking. Compared to traditional systems such as ELO-based matchmaking, this system provides enhanced capabilities through temporal performance weighting, disruptive player detection, and multi-level player classification.

The system's modular design ensures scalability and adaptability, while the iterative convergence mechanism guarantees stable skill ratings and fair rankings even in the presence of noisy or disruptive player behaviour.

VII. LIMITATIONS OF THE PROPOSED SYSTEM

While the proposed shooter game system demonstrates strong performance, several limitations must be considered for practical deployment:

1) Dependency on Server Infrastructure

The system relies on real-time backend servers for matchmaking, skill evaluation, and fair-play tracking. Server outages, high latency, or resource limitations may affect gameplay experience and scalability, especially in live multiplayer environments.

2) Potential for Misclassification of Player Skill

Player skill and behaviour may occasionally be misestimated due to noisy performance data, inconsistent participation, or rare gameplay scenarios. High-performing players may be undervalued, and low-performing players may be overrated, requiring monitoring and manual adjustments in some cases.

3) Sensitivity to Disruptive Behaviour

Although the system mitigates disruptive actions (AFK, toxicity, cheating), coordinated or extreme malicious behaviour may temporarily affect matchmaking fairness and skill ratings before iterative updates stabilize the system.

4) Limited Real-world Validation

Evaluation was conducted in a simulated environment with controlled player behaviour. Real-world deployment may introduce unpredictable factors such as network instability, unmodeled strategies, or large-scale player variability, requiring further testing.

5) Scalability and Parameter Dependency

System performance depends on parameters such as skill update rates, fair-play penalties, and convergence thresholds. Improper tuning may lead to slow adaptation, unstable rankings, or unfair matchmaking in large-scale or highly active game environments.

VIII. CONCLUSION AND FUTURE WORK

This paper presented a shooter game system designed to enhance skill-based matchmaking, fair-play enforcement, and competitive balance in multiplayer environments. The proposed approach integrates iterative skill and fair-play modeling, performance-based scoring, and behaviour-aware evaluation to effectively distinguish between reliable, high-performing players and disruptive participants. The system reduces the impact of toxic or AFK players while preserving meaningful contributions from support and strategic players, thereby improving overall match quality and player experience.

The evaluation results demonstrate that the system achieves high accuracy in skill prediction, stable convergence of rankings, and robustness against disruptive behaviour. Although the system is evaluated in a simulated environment, it highlights the potential of combining iterative ranking models

with behaviour-aware metrics for scalable and intelligent matchmaking in multiplayer games.

Future work will focus on extending the system to live, large-scale game deployments, incorporating advanced detection of coordinated disruptive behaviour, and improving adaptability through dynamic parameter tuning. Additionally, integration with AI-driven opponent bots, predictive analytics, or self-hosted scoring engines will be explored to enhance deployment flexibility and reduce external dependencies.

REFERENCES

[1] A. Herbrich et al., "TrueSkill™: A Bayesian Skill Rating System," Advances in Neural Information Processing Systems (NeurIPS), 2007.

[2] R. Cook et al., "Fairness and Anti-Cheat in Competitive Multiplayer Games," IEEE Transactions on Games, 2021.

[3] B. Silver et al., "Mastering the Game of Go with Deep Neural Networks and Tree Search," Nature, 2016.

[4] C. Hunicke et al., "AI for Dynamic Difficulty Adjustment in Multiplayer Games," Game AI Conference Proceedings, 2020.

[5] T. Balduzzi et al., "Robust Matchmaking in Online Competitive Games," ACM Computing Surveys, 2022.

[6] M. Cook et al., "Player Behaviour Analytics for Large-Scale Multiplayer Games," IEEE Access, 2023.

[7] Epic Games, "Unreal Engine Documentation and Networking Framework," Unreal Engine, 2024.