IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

Impact Factor 8.311 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 2, February 2026

DOI: 10.17148/IARJSET.2026.13248

Development of Interactive Games Using Unity Engine

Rooban R[1], Dr. K. Banuroopa[2]

Student, Department of Information Technology, Dr. N. G.P Arts and Science College, Coimbatore-641048, India[1]

Associate Professor, Department of Information Technology, Dr. N. G.P Arts and Science College, Coimbatore-641048, India[2]

Abstract: The gaming industry has seen remarkable growth over the past decade, driven by advances in game engines, programming languages, and interactive design principles. Unity Engine has emerged as one of the most popular and accessible platforms for developing both 2D and 3D interactive games. This paper explores the development process of interactive games using Unity Engine, with a focus on game object management, C# scripting, physics simulation, user interface design, and testing methodologies. The study presents a structured approach to game development, covering the complete lifecycle from conceptualization to deployment. A prototype game was developed to demonstrate the practical application of Unity's core features including scene management, animation systems, collision detection, and input handling. The results indicate that Unity provides an efficient and beginner-friendly environment for developing high-quality interactive games. The paper concludes with observations on the strengths and limitations of Unity for student-level game development projects and offers suggestions for future enhancements.

Keywords: Unity Engine, Game Development, C# Programming, Interactive Games, 2D 3D Games, Game Objects, Scripting, Game Testing

I. INTRODUCTION

The video game industry has evolved from simple pixelated graphics to complex, immersive virtual environments. Modern game development requires a combination of programming skills, artistic design, and an understanding of player interaction principles. Game engines serve as the foundational tools that enable developers to create interactive experiences without building everything from scratch.

Unity Engine, developed by Unity Technologies, is a cross-platform game engine that supports the development of both two-dimensional (2D) and three-dimensional (3D) games. It provides an integrated development environment (IDE) with built-in support for physics, rendering, scripting, audio, and networking. Unity uses C# as its primary scripting language, making it accessible to students and beginner developers who have prior exposure to object-oriented programming concepts.

The objective of this paper is to present a structured overview of the game development process using Unity Engine. The study covers the key aspects of Unity-based development including the use of game objects, prefabs, scripting in C#, physics components, user interface (UI) design, and testing. A prototype interactive game was developed as part of this study to demonstrate the practical implementation of these concepts.

This paper is organized as follows: Section 2 presents a review of related literature. Section 3 describes the methodology adopted for development. Section 4 discusses the results and observations. Section 5 provides the conclusion and future scope.

II. LITERATURE REVIEW

Several studies have explored the use of Unity Engine in educational and professional game development contexts. Craighead et al. (2008) examined the use of game engines for developing serious games and simulations, highlighting Unity's flexibility and rapid prototyping capabilities. Their work demonstrated that Unity could be effectively used for non-entertainment applications such as training simulations.

Messaoudi et al. (2015) conducted a comparative analysis of game engines including Unity, Unreal Engine, and CryEngine. Their findings indicated that Unity offered a balanced combination of performance, ease of use, and community support, making it particularly suitable for indie and educational game development projects.

IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

Impact Factor 8.311 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 2, February 2026

DOI: 10.17148/IARJSET.2026.13248

Dhillon and Kaur (2018) studied the application of Unity Engine for developing 2D mobile games. They reported that Unity's component-based architecture and extensive asset store significantly reduced development time compared to traditional coding approaches. The study emphasized the role of prefabs and reusable scripts in maintaining code quality. Politowski et al. (2021) surveyed the landscape of game testing methodologies and found that many small-scale game projects lacked formal testing procedures. They recommended the integration of unit testing and playtesting frameworks within game engines to improve software quality.

Fernandez (2019) explored the use of C# scripting in Unity for implementing game mechanics such as player movement, enemy behaviour, and scoring systems. The study provided practical examples of script organization and highlighted the importance of following object-oriented design principles in game development.

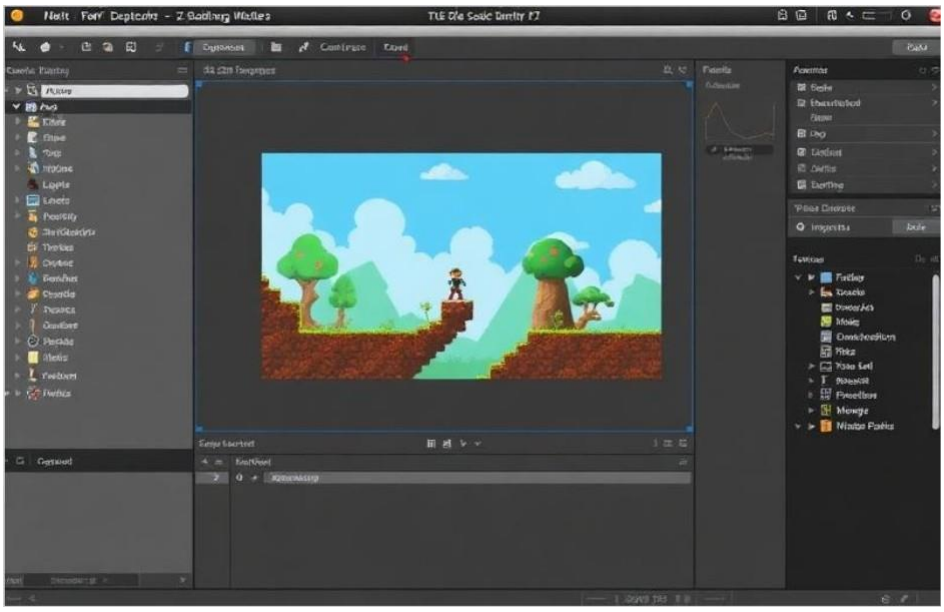

Fig 1: Game Environment Developed in Unity Editor

III. METHODOLOGY

The development process followed a structured software development lifecycle adapted for game development. The methodology consisted of five phases: requirement analysis, design, implementation, testing, and evaluation.

3.1 Requirement Analysis

The initial phase involved identifying the scope and objectives of the game. The target platform was defined as Windows desktop. The game was designed as a 2D side-scrolling adventure game where the player controls a character navigating through multiple levels while avoiding obstacles and collecting items.

3.2 System Design

The system architecture was designed using Unity's component-based model. Each interactive element in the game was represented as a Game Object with attached components for rendering, physics, and behaviour. The main components of the system included:

Player Controller — handles user input and character movement
Enemy Controller manages enemy behaviour and patrol patterns
Game Manager controls game state, scoring, and level transitions
UI Manager — manages health bars, score display, and menu screens
Audio Manager handles background music and sound effects

3.3 Implementation

The game was developed using Unity version 2022.3 LTS (Long Term Support). C# was used as the scripting language within the Visual Studio Code editor. The implementation phase covered the following key areas:

IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

Impact Factor 8.311 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 2, February 2026

DOI: 10.17148/IARJSET.2026.13248

Game Object Management: All visual and interactive elements were created as Game Objects in the Unity hierarchy. Prefabs were used to create reusable templates for frequently instantiated objects such as enemies, collectible items, and projectiles. This approach reduced redundancy and improved development efficiency.

C# Scripting: Custom scripts were written to control player movement, enemy artificial intelligence (Al), collision responses, and game flow. The MonoBehaviour class served as the base class for all game scripts, providing access to Unity's lifecycle methods including Start(), Update(), and FixedUpdate(). Input was captured using Unity's Input Manager, which mapped keyboard and controller inputs to in-game actions.

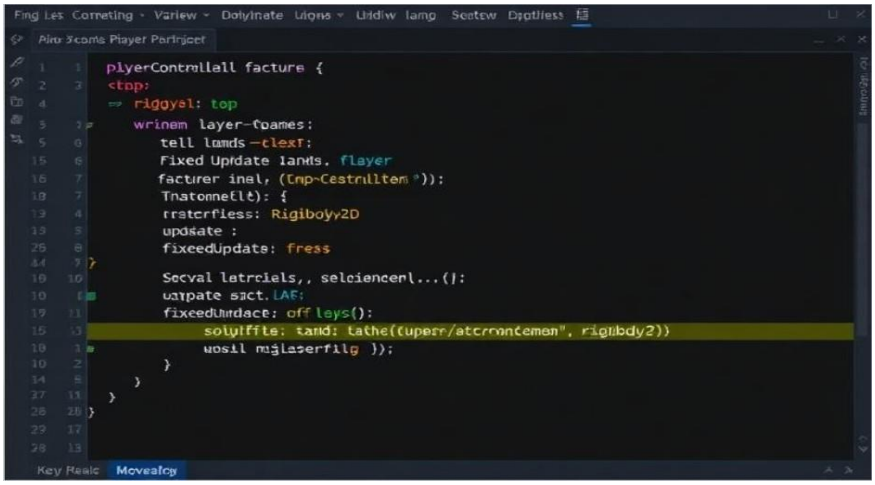

Fig 2: Player Control Script in C#

Physics and Collision Detection: Unity's built-in 2D physics engine was used to simulate gravity, forces, and collisions. Rigidbody2D components were attached to dynamic objects, while Collider2D components defined the physical boundaries of game objects. Trigger-based collision detection was implemented for collecting items and detecting level boundaries.

Animation: Character animations were created using Unity's Animator Controller and Animation window. Sprite sheets were used for 2D character animation, with transitions between idle, running, jumping, and attack states controlled through Animator parameters set via C# scripts.

User Interface Design: The game's user interface was built using Unity's Canvas system. UI elements including health bars, score counters, pause menus, and game-over screens were designed using the Retransform component. The UI was made responsive using anchor points and layout groups to adapt to different screen resolutions.

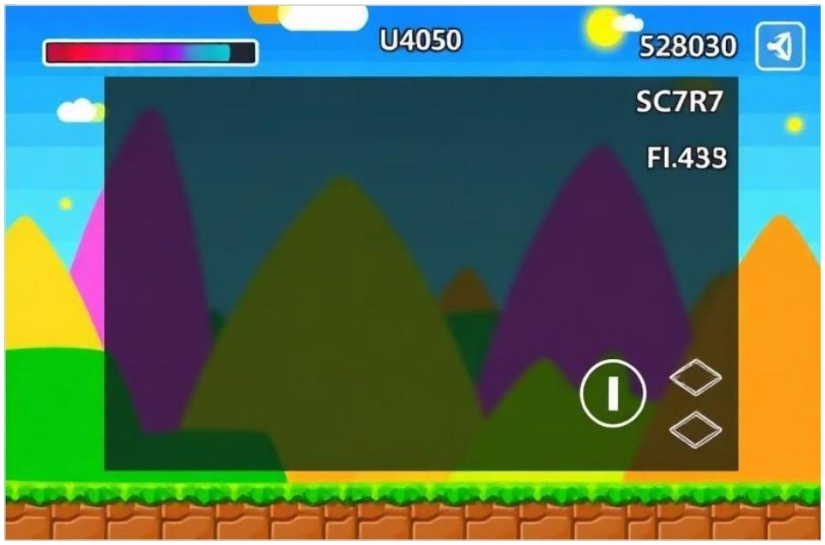

Fig 3: Game User Interface Layout

IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

Impact Factor 8.311 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 2, February 2026

DOI: 10.17148/IARJSET.2026.13248

3.4 Testing

Testing was conducted in multiple phases. Functional testing verified that all game mechanics worked as intended, including player movement, enemy behaviour, collision responses, and UI interactions. Performance testing assessed frame rate consistency and memory usage during gameplay. Playtesting sessions were conducted with a group of five student testers who provided feedback on game difficulty, controls, and overall experience.

Unity's built-in Profiler tool was used to monitor CPU usage, rendering performance, and memory allocation during test runs. Issues identified during testing were documented and resolved iteratively.

Table 1: Testin Parameters and Results

Table 1: Testing Parameters and Results

Test Phase	Parameter Tested	Tool / Method	Result / Observation
Functional Testing	Player Movement	Manual Playtesting	Smooth, responsive controls
Functional Testing	Enemy Behavior	Manual Playtesting	AI pathfinding worked correc
Functional Testing	Collision Detection	Unity Physics Engine	No clipping or missed collisic
Functional Testing	UI Interactions	Manual Playtesting	Menus and HUD responded c
Performance Testing	Frame Rate (FPS)	Unity Profiler	Stable 55–60 FPS during gan
Performance Testing	CPU Usage	Unity Profiler	Average 35% CPU usage
Performance Testing	Memory Allocation	Unity Profiler	No memory leaks detected
Performance Testing	Rendering Performance	Unity Profiler	Draw calls within acceptable
Playtesting	Game Difficulty	Student Feedback (n=5)	Rated as moderate; adjustm
Playtesting	Control Responsiveness	Student Feedback (n=5)	Positive response; minor twe
Playtesting	Overall Experience	Student Feedback (n=5)	Average rating: 4.2 / 5.0

IV. RESULTS AND DISCUSSION

The prototype game was successfully developed and tested on Windows desktop. The game included three playable levels with increasing difficulty, a scoring system, health management, and enemy encounters. The following observations were made during the development and testing process.

Unity's component-based architecture proved to be highly effective for organizing game logic. By attaching modular scripts to individual Game Objects, it was possible to develop, test, and modify individual game mechanics independently without affecting the overall system. This approach aligned well with object-oriented programming principles taught in the B.Sc. IT curriculum.

The C# scripting environment in Unity provided a familiar programing experience for students with prior knowledge of Java or The availability of comprehensive documentation and community tutorials further reduced the learning curve. However, certain advanced features such as shader programming and network multiplayer required additional expertise beyond the scope of this study.

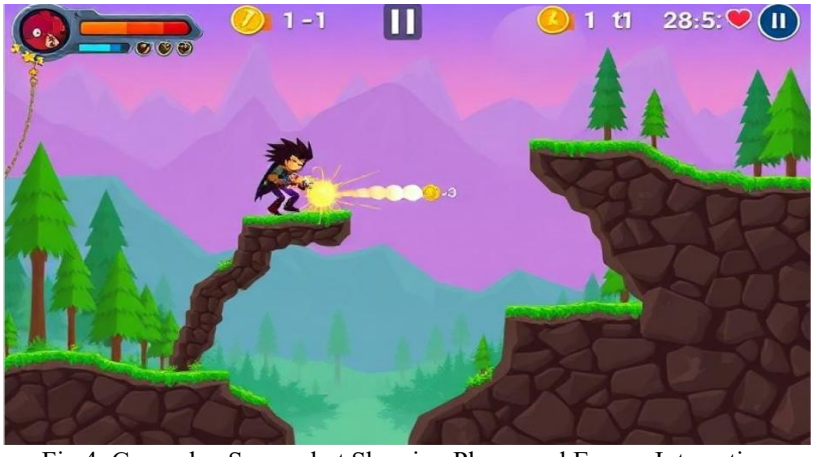

Fig 4: Gameplay Screenshot Showing Player and Enemy Interaction

IARJSET ISSN (O) 2393-8021. ISSN (P) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

Impact Factor 8.311 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 2, February 2026

DOI: 10.17148/IARJSET.2026.13248

The Unity Asset Store was utilized to source free graphical assets and sound effects, which accelerated the development process. However, reliance on external assets introduced consistency challenges in visual style, which were addressed through post-processing adjustments.

Performance analysis using the Unity Profiler revealed that the game maintained a stable frame rate of 55-60 frames per second (FPS) on a mid-range laptop computer with Intel i5 processor, 8GB RAM, and integrated graphics. Memory usage remained below 500MB throughout testing, indicating efficient resource management.

Feedback from playtesting sessions indicated that the game controls were intuitive and the difficulty progression was appropriate. Testers suggested the addition of a tutorial level and more varied enemy types as potential improvements for future versions.

Fig 5: Performance Analysis Using Unity Profiler

V. CONCLUSION

This paper presented a structured approach to developing interactive games using Unity Engine. The study demonstrated that Unity provides a comprehensive and accessible platform for game development, particularly suitable for students and beginner developers. The component-based architecture, combined with C# scripting capabilities, enables rapid prototyping and iterative development.

The prototype game developed as part of this study successfully incorporated key game development concepts including game object management, physics simulation, animation, UI design, and testing. The testing results confirmed that the game performed reliably across all functional and performance parameters.

The study also identified areas for future improvement, including the integration of multiplayer networking, implementation of advanced Al behaviours using state machines, and optimization of assets for mobile platform deployment. Additionally, the incorporation of virtual reality (VR) support using Unity's XR toolkit represents a promising direction for future research.

In conclusion, Unity Engine serves as an effective tool for learning and applying game development concepts within an academic context. The skills acquired through Unity-based projects are transferable to professional game development and related fields such as simulation, visualization, and interactive media.

REFERENCES

[1]. Craighead, J., Burke, J., & Murphy, R. (2008). Using the Unity Game Engine to Develop SARGE: A Case Study. Proceedings of the Workshop on Simulation, Training, and Education.
[2]. Messaoudi, F., Simon, G., & Ksentini, A. (2015). Dissecting Games Engines: The Case of Unity3D.
[3]. International Workshop on Network and Systems Support for Games (NetGames), pp. I -6.
[4]. Dhillon, H. K., & Kaur, A. (2018). A Comprehensive Study of 2D Game Development using Unity Engine. International Journal of Computer Sciences and Engineering, 6(7), pp. 543-548.

IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

Impact Factor 8.311 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 2, February 2026

DOI: 10.17148/IARJSET.2026.13248

[5]. Politowski, C., Petrillo, F., & Guéhéneuc, Y. G. (2021). A Survey of Video Game Testing. IEEE Transactions on Games, 14(3), pp. 440-452.
[6]. Fernandez, A. (2019). C# Scripting in Unity: Best Practices for Game Developers. Journal of Game Development and Education, 3(1), pp. 12-22.
[7]. Goldstone, W. (2011). Unity 3.x Game Development Essentials. Packet Publishing.
[8]. Hocking, J. (2018). Unity in Action: Multiplatform Game Development in C#. Manning Publications.
[9]. Juliani, A., Berges, V. P., Teng, E., Cohen, A., Harper, J., Elion, C., & Lange, D. (2020). Unity: A General Platform for Intelligent Agents. arXiv preprint arXiv: 1809.02627v2.
[10]. Okita, A. (2014). Learning C# Programming with Unity 3D. CRC Press.
[11]. Unity Technologies. (2023). Unity Documentation. Retrieved from https://docs.unity3d.com/

