Industry Science and Engineering Vol. 2 No. 1, 2025

Design of the Top-Level Code Framework for Unity Based on C# Language

Qi Zhang, Ning Mao, Jinxiu Li, Ningning Zhou, Jingyi Miao, Guwei Li *
Artificial Intelligence College, Zhejiang Dongfang Polytechnic, Wenzhou, Zhejiang, China
**Corresponding Author.*

Abstract: This paper focuses on exploring the design of a top-level code framework for Unity based on the C# language. As a popular game development engine, Unity is widely used in various fields such as virtual reality and gaming, thanks to its cross - platform capabilities and rich resources. The C# language, with its simplicity, powerful functionality, and object - oriented features, is closely integrated with Unity and has become the key to building efficient applications. Through in - depth analysis of design principles, construction of core modules, and application of real - world cases, this paper presents a complete and practical top - level code framework, which provides strong support for improving the development efficiency and quality of Unity projects.

Keywords: Unity; C#; Framework; Code

1. Introduction

Since its launch, the Unity engine has quickly gained prominence in the fields of game development, virtual reality (VR), augmented reality (AR), and other interactive applications. Its cross - platform nature enables developers to easily deploy projects to a variety of devices, from desktop computers to mobile devices, and even emerging VR headsets. As the complexity of applications increases, efficient code organization and architecture design become increasingly important. The C# language, as the main scripting language for Unity, has a concise syntax, strong type safety, and a rich class library, and it has significant advantages in building robust and maintainable code. Designing a reasonable top - level code framework helps standardize the code structure in Unity projects, improve development efficiency, enhance code extensibility and reusability, and thus promote innovation and development in related fields. Foreign countries are at the forefront of research on the combination of Unity and C# in development. Many large game companies and research institutions have conducted in - depth research on optimizing the C# - based Unity code framework for high - complexity games and professional simulation applications. In terms of performance optimization, they use advanced algorithms and data structures to improve rendering efficiency and memory management capabilities. For example, some AAA - level games optimize resource loading and scene switching using C# features during development to achieve a smooth gaming experience. In China, with the development of the game industry and VR/AR technologies, research on Unity development has also deepened. Universities and enterprises focus on combining domestic market demands to explore framework designs suitable for different application scenarios, such as the implementation of interactive logic and optimization of user experience in educational and medical applications. However, overall, there is still room for further improvement in the research on a general and efficient top - level code framework.

The main research contents of this paper include: in - depth analysis of the characteristics and advantages of the C# language in the Unity development environment, clarification of the design principles and key points of the top - level code framework suitable for Unity projects; construction of a complete C# - based top - level code framework for Unity, covering the design and implementation of core modules; verification of the framework's effectiveness through real - world cases, summarization of experience, and proposal of improvement directions. In terms of research methods, the literature review method is used to sort out relevant research results at home and abroad; the case analysis method is used to analyze the code architectures of successful Unity projects; and combined with theoretical derivation, a scientific and reasonable framework system is constructed from the perspectives of software engineering and programming language characteristics.

2. Basics of C# Language and Unity Development

2.1 Advantages of C# Language Features in Unity

The C# language was developed by Microsoft. Its concise syntax reduces the difficulty of code writing. In Unity development, developers can quickly get started with writing various scripts. For example, when defining variables and methods, the syntax is clear and intuitive, reducing redundant code. Type safety ensures that type errors are detected during compilation and runtime, improving code stability [1]. Especially in complex game logic and data interaction, it effectively avoids program crashes caused by type mismatches. C# fully supports object - oriented programming. The encapsulation feature encapsulates data and operations in classes. For example, a game character class contains data such as health points and attack power, as well as methods such as attack and movement, enhancing code security and maintainability. The inheritance mechanism allows sub - classes to reuse the attributes and methods of the parent class. For example, different types of enemies can inherit from a general enemy class, reducing duplicate code. Polymorphism enables objects of different classes to respond differently to the same message. For example, the attack methods of different weapon classes achieve different attack effects, enriching the gameplay. In addition, C#'s powerful class library covers file operations, network communication, graphics processing, etc. In Unity projects, functions such as resource loading, online battles, and interface drawing can be easily implemented, greatly improving development efficiency [2].

2.2 Overview of the Unity Engine and Its Development Modes

The Unity engine provides a visual development environment, including tools such as the scene editor and resource manager, which facilitate developers to create and manage game scenes, characters, and other resources. It supports multiple development modes, such as the component - based development mode. A game object is composed of multiple components, and each component is responsible for a specific function. For example, the Transform component controls position, rotation, and scaling, and the Rigidbody component implements physical simulation. This mode makes the code modular, facilitating reuse and maintenance. Unity also has powerful cross - platform capabilities. Developers only need to write the code once and can deploy it to different platforms, such as Windows, Mac, iOS, Android, and VR devices, through simple settings, saving development time and costs. In terms of rendering, Unity uses an advanced rendering pipeline and supports real - time rendering of high - quality 3D scenes, providing users with a realistic visual experience. At the same time, Unity has a large asset store, where developers can obtain various pre - made resources and plugins to further accelerate the development process.

2.3 Collaborative Working Mechanism of C# and Unity

In Unity, C# scripts are the core for controlling game logic [3]. Developers write C# scripts and attach them to game objects to control the behavior of game objects. For example, a C# script can be written to control the movement of a character, obtain user input (such as keyboard and joystick operations), and modify the properties of the character's Transform component to achieve the character's movement in the scene. C# scripts are closely integrated with Unity's component system and can access and operate the properties and methods of components. At the same time, Unity provides a rich set of API interfaces, and C# scripts can call these interfaces to implement various functions, such as loading scenes, playing sound effects, and creating special effects [4]. In terms of resource management, C# code can load resources such as models, textures, and audio from project resources through Unity's resource loading mechanism and dynamically instantiate and manage them at runtime. In addition, Unity's event system collaborates with C#'s delegate and event mechanisms to handle various events in the game, such as collision detection and UI interaction, making the game logic more flexible and efficient.

3. Design Principles of the Unity Top - Level Code Framework

3.1 Maintainability Principle

The framework structure should be clear and well - defined, with different functional modules clearly divided. For example, functions such as game logic, user interface, and data management should be placed in different modules [5]. The code within each module should follow a unified coding standard and have good comments, making it easy for developers to understand and modify. Design patterns, such as the singleton pattern for managing global data and the factory pattern for creating objects, should be used to make the code structure more reasonable and reduce the maintenance difficulty. At the same time, the coupling degree between modules should be reduced so that when the function of one module is updated or modified, it will not affect the normal operation of other modules as much as possible.

3.2 Extensibility Principle

Considering the changing project requirements and function upgrades, the framework should reserve extension interfaces and space. For example, in the game character system, if new character types may be added in the future, the character base class should be designed to be extensible, and new character classes can inherit from the base class and add unique functions [6]. A plug - in architecture should be adopted to facilitate the integration of new functional modules later, such as adding new game play modules or third - party service plugins, without the need for large - scale modification of the existing code structure.

3.3 Performance Optimization Principle

In Unity development, performance is crucial. The framework should use efficient data structures and algorithms to reduce memory usage and computational overhead. For example, the object pool technology should be used to manage objects that are frequently created and destroyed, such as bullets and monsters in the game, to avoid repeated memory allocation and deallocation. The rendering code should be optimized, the rendering levels of objects should be set reasonably, and the occlusion culling technology should be used to reduce unnecessary rendering. For complex calculations, asynchronous processing should be adopted to avoid main - thread freezes and ensure smooth game operation.

3.4 Compatibility Principle

Since Unity supports multiple platforms, the framework design must ensure stable operation on different platforms. When writing code, the principle of platform independence should be followed, and specific platform APIs should be avoided unless necessary. For differences in platform characteristics, such as input devices and display resolutions, the framework should provide a unified interface for adaptation. In terms of resource management, it should be ensured that resources are correctly loaded and displayed on different platforms to guarantee a consistent gaming experience across all platforms.

4. Design of Core Modules of the C# - Based Unity Top - Level Code Framework

4.1 Scene Management Module

This module is responsible for loading, unloading, and switching game scenes. Using Unity's SceneManager class, asynchronous scene loading is implemented through C# scripts to avoid game freezes during the loading process [7]. At the same time, it manages the generation and destruction of objects in the scene and maintains the state of scene objects. A SceneObjectManager class is created to uniformly manage the lifecycle of objects in the scene. For example, when the scene is switched, objects that are no longer needed in the current scene are automatically destroyed to release memory resources.

4.2 Character Control Module

This module implements functions such as the movement, animation playback, and state management of game characters. By detecting user input (such as keyboard, joystick, and touch operations), the movement of the character is controlled [8]. Combining the Animator component with C# scripts, the character's animation switching is realized. According to the character's state (such as walking, running, jumping, and attacking), the parameters of the Animator are controlled through C# code to play the corresponding animations. At the same time, it manages the character's attributes, such as health points and magic points. When the character is attacked or uses a skill, the attribute values are updated, and other relevant modules are notified through the event system.

4.3 Interaction Logic Module

This module handles the interaction between the user and the game world, including UI interaction and physical interaction. In terms of UI interaction, using Unity's UGUI system, responses to operations such as button clicks, swipes, and drag - and - drops are implemented through C# scripts. In physical interaction, object collisions and trigger events are detected, and the corresponding logic is implemented through C# code [9]. For example, when the character collides with an item, the item pickup logic is triggered, and the character's item inventory data is updated. For complex interactions, such as gesture recognition in VR interaction, third - party libraries (such as the Leap Motion SDK) can be integrated, and the library interfaces can be connected through C# code to implement gesture control functions.

4.4 Data Management Module

This module is responsible for the storage, reading, and updating of game data. For static data, such as game configurations and initial character attributes, it can be stored in files in formats such as XML and JSON, and the data is read through C#'s file reading classes. For example, JSON files can be used to store game level information, which is read through code. For dynamic data, such as the player's game progress and scores, a database can be used for storage. In Unity, a lightweight database such as SQLite can be selected, and data reading and writing are implemented through C#'s database operation classes (such as System. Data. SQLite). At the same time, data consistency and synchronization between different modules should be ensured. The event mechanism is used to notify relevant modules of data updates. For example, after the player completes a task, the task data is updated, and the UI module is notified to display the new task status.

5. Framework Implementation and Optimization

5.1 Inter - Module Communication Mechanism

To achieve effective communication between modules, an event - driven and message - passing mechanism is adopted. In C#, through the definition of delegates and events, loosely coupled communication between modules is realized. For example, after the scene management module finishes loading a scene, it triggers an event. The character control module and the interaction logic module can subscribe to this event and perform corresponding operations when the event is triggered, such as initializing the character and loading UI elements. At the same time, a message center class is created to manage and distribute various messages. Each module sends messages to the message center, and the message center distributes the messages to the interested modules. For example, when the character control module sends a character death message, the message center notifies the data management module to record the game result and the UI module to display the game over interface.

5.2 Code Optimization Techniques

During the C# code writing process, a series of optimization techniques are adopted to improve performance. Complex calculations should be avoided in high - frequency call functions such as Update. Data that can be cached should be calculated and cached in advance [10]. For example, when calculating the character's movement direction, if the direction remains unchanged, the calculation result can be cached to reduce repeated calculations. Boxing and unboxing operations should be reduced. When handling conversions between value types and reference types, type compatibility should be noted to avoid unnecessary boxing and unboxing overhead. Generics should be used to improve code reusability and performance. For example, a general object pool class can be created for managing objects of different types. At the same time, Unity's Profiler tool should be used regularly to analyze code performance, identify performance bottlenecks, and optimize them accordingly.

5.3 Resource Management Strategy

In Unity projects, resource management has a significant impact on performance. Resource packaging and compression techniques are adopted to reduce the size of resource files and speed up the loading process. Unity's AssetBundle function is used to package resources into AssetBundle files, which are loaded on demand at runtime. For texture resources, the compression format and resolution should be set reasonably to reduce memory usage while ensuring the visual effect. At the same time, a resource caching mechanism should be established. Frequently used resources, such as common UI icons and sound effects, should be cached in memory to avoid repeated loading. After the resources are loaded, resources that are no longer used should be released in a timely manner. Through methods such as resource reference counting, effective memory resource recovery is ensured.

6. Detailed Design Case of the Framework

6.1 Project Background and Objectives

This case is a virtual reality adventure game developed based on Unity. The game is set on a mysterious island. The player controls a character to explore the island, solve puzzles, fight monsters, and complete various tasks. The project objective is to create an immersive VR gaming experience with smooth performance, rich interactions, and an engaging storyline.

6.2 Framework Application Process

6.2.1 Scene building and management

Using the scene management module of the framework, multiple island scenes, including forests, caves, and beaches, are created according to the game design. Asynchronous loading technology is used to ensure smooth scene switching. Various objects, such as trees, rocks, and buildings, are arranged in the scene. The SceneObjectManager class is used to manage the generation and destruction of objects. For example, after the player leaves an area, objects that are no longer needed in that area are automatically destroyed to optimize memory usage.

6.2.2 Character control and interaction

The character control module is used to implement actions such as the movement, jumping, and climbing of the player character in the VR environment. Combined with the input of VR devices such as the HTC Vive, the character's behavior is precisely controlled. Through the interaction logic module, interaction with objects in the scene, such as picking up items, opening doors, and solving puzzles, is realized. Gesture recognition technology (integrating the Leap Motion SDK) is used to allow the player to operate objects through gestures, enhancing the realism of the interaction.

6.2.3 Data management and storyline progression

With the help of the data management module, data such as the player's game progress, collected items, and completed tasks are stored. The SQLite database is used to record the data, and the player can continue the game when logging in next time. According to the storyline design, the game state is updated through the data management module, and different storyline events are triggered. For example, after the player completes a specific task, a new scene and storyline are unlocked.

6.3 Project Results and Problem Solving

By applying the C# - based Unity top - level code framework, a virtual reality adventure game was successfully developed. During the testing process, the game showed good fluency and interaction experience and received positive feedback from players. However, some problems were also encountered during the development process. For example, in complex scenes, rendering freezes occurred. After analysis, it was found that the problems were related to lighting calculations and object rendering order. By optimizing the lighting settings, using baked lighting to reduce real - time lighting calculations, and adjusting the object rendering levels, the freeze problem was solved. In addition, in the development of the online multiplayer function, data synchronization problems were encountered. By improving the synchronization mechanism of the data management module and using a reliable network communication protocol (such as UDP), data consistency among players in the online game was ensured, and a stable online experience was achieved. By solving these problems, the practicality and extensibility of the framework were verified, and experience was accumulated for subsequent projects.

7. Conclusion

This paper successfully designed and implemented a C# - based top - level code framework for Unity. Through in - depth analysis of the characteristics of the C# language and the development requirements of Unity, the design principles of the framework were established, and a framework system covering core modules such as scene management, character control, interaction logic, and data management was constructed. Through real - world case verification, this framework effectively improves the development efficiency of Unity projects, enhances code maintainability, extensibility, and performance. In the case development, problems such as rendering freezes and network data synchronization were successfully solved, proving the effectiveness of the framework in dealing with complex project challenges. As the Unity engine continues to be upgraded and new technologies emerge, the C# - based Unity top - level code framework has broad development prospects. At the technical level, with the improvement of hardware performance, the framework can be further optimized to support more complex scenes and special effects, such as real - time global lighting and high - resolution texture processing. In the application field, with the popularization of VR/AR technologies, the framework can be extended to more industry applications, such as medical training and industrial design simulation, and the interaction logic and data management can be optimized according to the needs of different industries. At the same time, by combining with artificial intelligence technologies, such as introducing AI algorithms in character behavior control and scene generation, the intelligence and innovation of the game can be improved. In addition, continuous attention should be paid to cross - platform compatibility to ensure the stable operation of the framework on emerging devices and platforms, providing developers with more powerful and flexible development tools.

References

[1] Ke Wang. Leveraging Unity for 2D Pixel Game Development: Techniques and Best Practices. ITM Web of Conferences, 2025, 70(1):3002-3002.
[2] Peter Snow. Virtual reality and pain management: The need for clarity for future interventions. British Journal of Pain, 2025, 19(2):68-70.
[3] Škola Filip, Boskovic Dusanka, Rizvic Selma, et al. Assessing User Experience and Cognitive Workload in Virtual Reality Digital Storytelling. International Journal of Human–Computer Interaction, 2024, 40(6):1479-1486.
[4] W. Fang, F. Zhang, Y. Ding, et al. A new sequential image prediction method based on lstm and dcgan. Computers, Materials & Continua, 2020, 64(1):217-231.
[5] Smith. Game architecture patterns in Unity. Journal of Game Development, 2021, 15(2):45-67.
[6] Y. Dai, Z. Luo. Review of unsupervised person re-identification. Journal of New Media, 2021, 3(4):132-133.
[7] H. Wang, L. Chen, A comparative study of game development frameworks. Computer Games Technology Review, 2021, 8(3):22-25.
[8] Nakamura. Advanced C# techniques for Unity developers. Game Programming Gems, 2020, 9(1):77-79.
[9] Johnson. Memory management in Unity: Best practices. Game Developer Magazine, 2021, 45(6):30-35.
[10]Ericson. Real-time game networking architectures. Proceedings of GDC, 2019, 3(2):1-5.