Received: 26 February 2023 | Revised: 10 May 2023 | Accepted: 4 August 2023

DOI: 10.1002/cav.2206

RESEARCH ARTICLE

WILEY

An auxiliary development framework for lightweight RPG games based on Unity3D

Bo Zhang | **Huiping Shi** | **Xinyu Wang**

Department of Computer Science, Jinling Institute of Technology, Nanjing, China

Correspondence
Bo Zhang, Department of Computer Science, Jinling Institute of Technology, No.99 Hongjing Avenue, Nanjing, 211169, Jiangsu, China.
Email: 30719547@qq.com

Funding information
Jinling Institute of Science Technology Research Initiation Project, Grant/Award Number: jit-b-202030

Abstract

With the growing prevalence of virtual reality technology across industries like cinema, video animation, and gaming, game developers must cater to a wider range of players. Game engines serve as middleware, reducing the time spent on coding and improving productivity for creating video games. While mainstream game engines offer a plethora of customization options, certain developers may find themselves limited when it comes to controlling specific aspects of the game. Due to the lack of built-in functionality in current game engines like Unity and Unreal, the author developed TRPGFramework, a lightweight framework based on Unity3D, to address some of these problems. The game framework consists of two main parts: ZGamework (overall framework) and BGamework (game-specific framework). To enhance performance, scalability, and code reusability in the game development process, the entity-component-system structure is implemented. This approach increases maintainability and makes the code clearer, easier to expand, and debuggable. By creating a simple 2D RPG game for performance testing, we confirmed that the framework significantly improves code reusability and modularity, resulting in faster development time and increased efficiency for interactive applications and games. This framework is particularly useful for creating real-time combat role-playing games, which are some of the most complex games available today. Using entity-component-system principles, we developed a buff system and unit controller optimized to allow hundreds of intelligent units to fight on the same screen or thousands of non-intelligent units to fight simultaneously.

KEYWORDS

entity-component-system, Unity3D, TRPGFramework

1 | INTRODUCTION

Model-view-controller (MVC), model-view-presenter (MVP), and model-view-view-model (MVVM) are three popular architectural styles used in video game development.[1] Their structures and interactions are depicted in Figure 1,

Huiping Shi and Xinyu Wang contributed equally to this work.

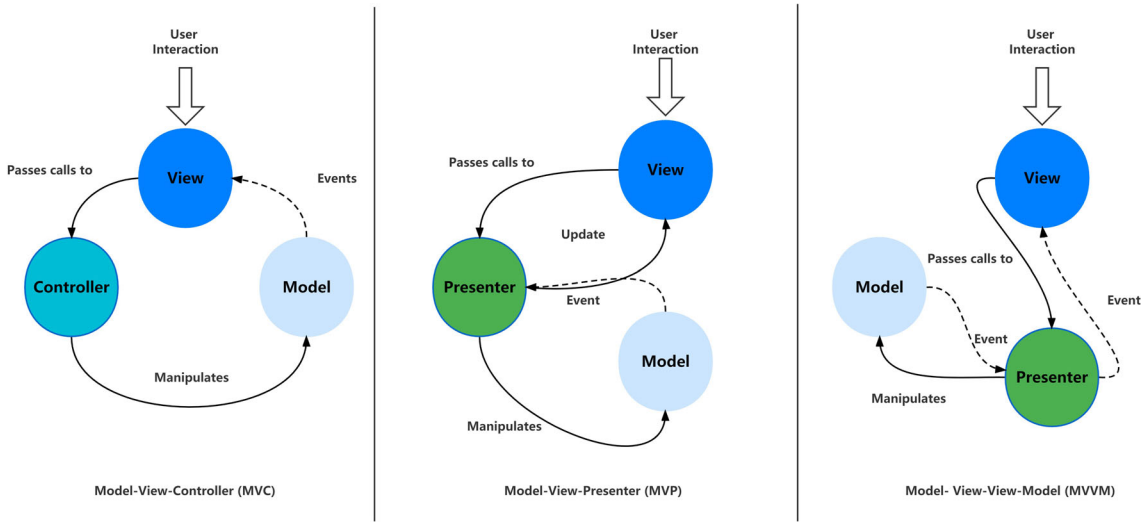


FIGURE 1 MVC, MVP, and MVVM architecture

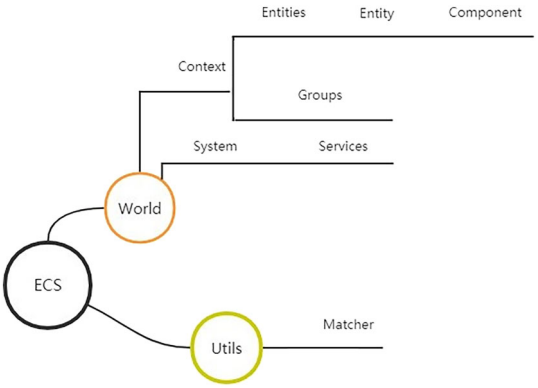


FIGURE 2 ECS architecture

which reveals that each framework possesses its unique advantages and potential uses.[2,3] While developers often use object-oriented methods, this can lead to maintenance and scalability issues as the game's features and versions increase.[4]

The entity-component-system (ECS) architecture comprises abstract business modules that serve a specific collection of components instead of managing entities. If an entity has this collection, it receives the services offered by the module. As illustrated in Figure 2, the ECS architecture defines each unit in a game, such as a monster or a camera, as an entity composed of one or more components.[5,6] Each component only contains the necessary data to perform its function; for instance, skill components hold skill damage and range. The system processes the collection of entities and holds only the logic, not the state, like a tool. For example, the skill system executes skills based on the state of each entity that has access to that state. This structure enables changing the behavior of reality in real-time by adding or removing entity parts.[7]

The Unity3D implementation layer acts as a bridge between the core layer of the ECS framework and practical game development. It provides a tangible realization of the ECS architecture for the Unity engine.[8,9] To allow developers to modify data easily in the editor, the core classes of the Unity implementation layer must be mountable on game objects.[10,11] The ECS framework includes an editor that simplifies manipulation of game data for non-developers. This editor can be used directly in the game scene or on a premade body. To achieve this, the three essential classes of the Unity implementation layer—EntityBehavior, ComponentBehavior, and SystemBehavior—all inherit from MonoBehavior. These classes' initialization methods use the inject keyword to leverage dependency injection capabilities.[12,13]

The ECS architecture and its variations, originating from the gaming industry, follow the composition over inheritance principle, which provides developers with more flexibility in defining simulation objects than conventional approaches.[14] To manage logical objects, each object must be assigned an entity behavior by the developer. Although entities are unique

throughout the program, each entity must have a corresponding entity behavior in Unity. In rare cases, a collection of in-game entities may represent a single "entity." Therefore, multiple entity behaviors with references to the same entity are possible.[15,16] One of these entity behaviors acts as the entity's body, while the proxy property of the other entity behavior maintains a reference to this body. Additionally, the transform component of the game object that contains these entity behaviors is kept in view of this entity. The entity's ViewComponent stores the transform component of the game object on which the entity behavior resides in its transforms property.[17] If a view component is not already attached to the game object, the entity behavior generates one. When an entity is destroyed, the disposer property removes any unwanted data streams and unsubscribes from them. The transforms property is a responsive container that stores the transform properties of all the game objects that comprise the entity, allowing access to all of its components.[18,19]

The current level of production has garnered a large audience, indicating a significant market opportunity.[20,21] However, popular game engines like Unreal Engine and Unity3D use the entity-component (EC) architecture, which does not support asynchronous and event-driven systems.[22] In contrast, the ECS architecture is specifically designed to meet these requirements by compartmentalizing data and behavior into separate layers. This article presents a comprehensive implementation of all the systems and frameworks necessary for creating a moderately sized game, with a strong emphasis on optimizing speed. Creating a lightweight RPG game requires not only excellent story design but also superb math and performance planning. The TRPGFramework draws inspiration from several existing frameworks, such as GameFramework, QFramework, and EGamePlay, and incorporates various features, such as compiler development, hierarchical property manipulation, object pooling, UI management, timer management, event management, and resource management. This framework is not limited to RPG games and can be applied to any game. The content is simplified to enable developers to quickly grasp and utilize it. Although it may not be as robust as other frameworks, it offers fast and convenient usage.[23,24]

2 | RELATED WORKS

The initial Unity game framework focused mainly on fundamental aspects of game development, such as object pooling, resource management, interface management, and event management. While useful, these frameworks did not offer enough support for advanced features and performance optimization in game development. For example, ymh's CatLib was released as an open-source game client based on Unity3D and features a server-side dual-end framework.[25] The server-side of the framework is constructed using C.net Core as a distributed game server, and it stands out for its high development efficiency, high performance, shared logic code between the client and server ends, client-side server hot-plugging mechanism, WebSocket protocol support, and various other features. As the game market continues to evolve, Unity-based game frameworks have been adapting to meet the changing needs of the market. For instance, in recent years, with the growing mobile game market, many Unity game frameworks have shifted their focus towards mobile platform optimization and adaptation, offering more features and tools suitable for mobile platforms. In 2020, Ellan Jiang's GameFramework encapsulated commonly used modules in the game development process, largely standardizing the development process, accelerating development, and ensuring product quality. Later, modules such as Config, Data Node, and Data Table were developed for use by game developers. This game framework is more focused on data flow processing, and the structure is more dispersed.

Xie'sQFramework is a rapidly evolving collection of frameworks designed to accommodate upgrades and changes in Unity software in 2021. Its goal is to become the first framework for companies without framework experience, independent developers, and Unity3D newcomers. The framework comprises several projects with solutions in various technical directions. The QFramework system design architecture is divided into four layers, each with its set of rules. The performance layer has the ViewController layer, which is responsible for receiving input and changing the state when performance changes, with the IController interface. QFramework is a game development framework that follows the MVC design pattern. In this pattern, developers need to write more code to facilitate the interaction between the controller and view, which increases the complexity of the code and maintenance costs. In certain scenarios, the relationship between the view, model, and controller is tightly coupled, making the code difficult to test and maintain. In Figure 3, we see a simple project based on the MVC architectural pattern. The model, view, and controller are organized in the inner layer. However, this layering can add complexity to the entire project, and the relationships between the models, views, and controllers can become tightly coupled, making it difficult to modify one component without affecting the others.

As the Unity engine evolves and the game market changes, game frameworks are becoming increasingly convenient and supportive for developers. In this article, the author drew inspiration from existing frameworks such as GameFramework and QFramework to create a simplified TRPGFramework that is easy to understand and use. One of the

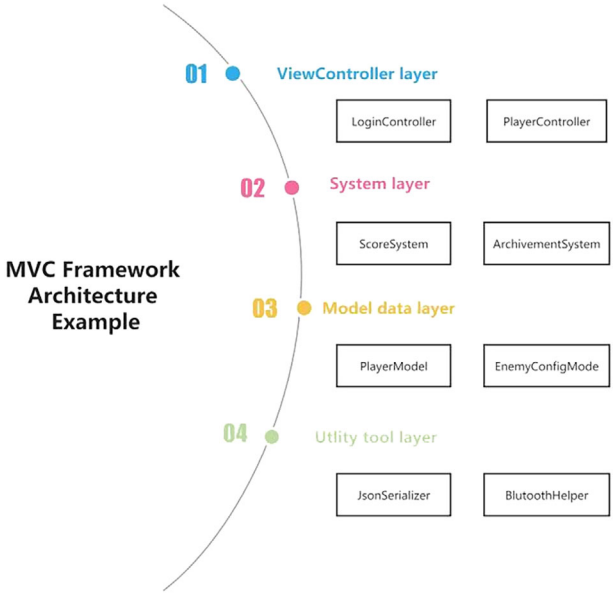


FIGURE 3 MVC framework architecture

distinguishing features of this framework is its use of an ECS architecture, which is lightweight and efficient. The authors incorporated the concept of semantic traits into their ECS, enabling compile-time checks, detecting entity class affiliation, and providing run-time functionality changes. To refine hierarchical management, TRPGFramework has been divided into two parts: the generic framework ZGamework, which includes lightweight development aids such as an event system, object pool, time manager, multi-level properties, and resource manager. This level's functionality is similar to the behavior layer of the ECS structure, as it processes entities through systems to provide them with corresponding behaviors. The systems are at the core of this level and perform specific actions on specific components. The specialized event framework BGamework includes a buff system, skill system, backpack system, combat system, AI system, and developer model, among other features. BGamework manages game object entities and their components, presents them to the user, and handles user input. These two frameworks combine to form a mature RPG game framework that improves development efficiency, code speed, code reusability, and modularity for interactive applications and games.

3 | FRAMEWORK ARCHITECTURE

3.1 | Design ideas

The TRPGFramework, constructed based on ECS, allows for the simulation of different types of games in Unity3D. In this article, the ECS framework's entity layer, behavior layer, and presentation layer are further divided into two parts: ZGamework and BGamework. The underlying game is based on ZGamework, and all repetitive objects are stored in an object pool. The different modules communicate with each other through an event system, and a timer is used to synchronize all time-related events. BGamework is used for game content management, map mapping using TileMap, and backpack management through BagManager. To ensure unified control of multiple types of items, all items inherit from UseBase. This approach can assist developers in having better control over game entities, handling game logic and data more effectively, and managing various events and behaviors within the game. Furthermore, Hierarchical management approach can provide better code organization and scalability, allowing developers to modify and add game features more easily to meet the ever-changing market demands.

The ZGamework is responsible for managing the input in each frame, which includes activating systems, registering events, UI windows, and object pools, as shown in Figure 4. Additionally, data in StreamingAssets can be read and initialized into multiple tables. As the interface framework entry of the ECS architecture, ZGamework implements various functions such as activating systems, registering events, UI windows, and object pools. The data in StreamingAssets is also read and initialized into multiple tables. The BGamework includes the initialization of the backpack system, the buff

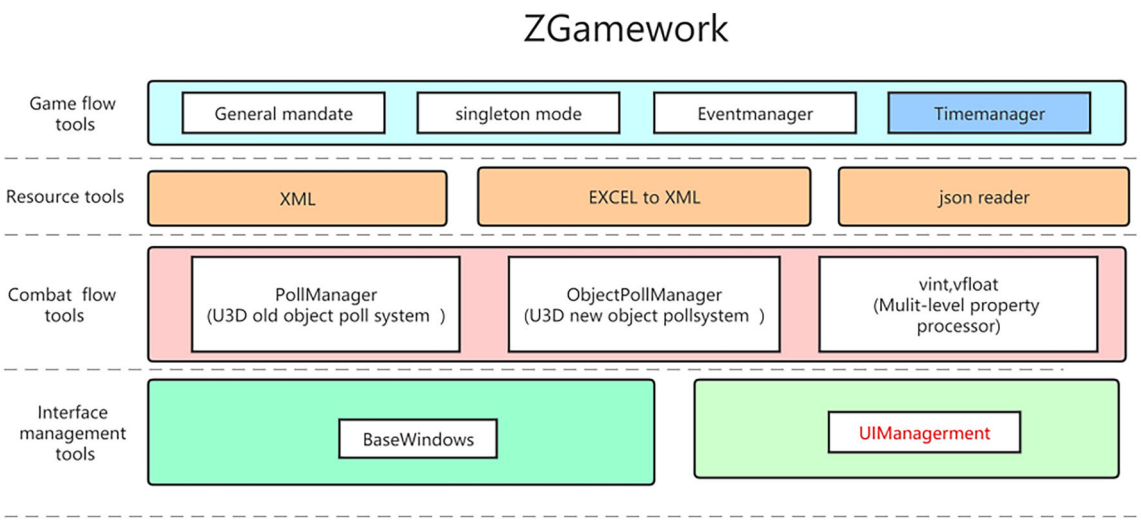


FIGURE 4 ZGamework structure

system, the enemy controller, and certain aspects of the game scene. By modeling data using XML-JSON, players have full control over the game data through the open database and StreamingAssets. Players can directly modify the database in StreamingAssets to impact the game content directly.[26]

3.2 | Data control aspect

3.2.1 | Data structure

To achieve full control over game data through an open database and StreamingAssets, traditional game frameworks typically use the XmlDocument or XmlReader class for data parsing. For example, in QFramework, we can utilize the XML utility class to read and parse XML documents. In this article's framework, we use LitJSON, which is a small library class with a low memory footprint that's perfect for use in performance-focused Unity projects. LitJ-SON is optimized for performance and can handle large JSON files quickly and efficiently. Using this approach, players can directly modify the database within StreamingAssets to influence game content. Developers can easily parse JSON format data and convert it into various data types in Unity using LitJSON.

As shown in Figure 5, we utilized LitJSON to establish a standardized process for serializing combat, which includes serialized maps and serialized combat units. This is achieved by using the ToObject method of the JsonMapper class to convert the JSON string into a JsonData object, which allows access to the individual fields of the JSON data. As a result, they can design and create many mods that they can plug into the game. Though this method completely foregoes security when compared to using a database like SQLite, it opens up the game's creative possibilities to the player.

3.2.2 | Behavior tree

In Unity software, the development time can be reduced by using the basic behavior tree provided by AI Designer and designing only the nodes of the behavior tree. The essence of a behavior tree includes nodes for options, sequences, conditions, events and so on. Code inherits various nodes and merges them into the tree, traverses the nodes from the root node, and performs AI actions based on a complex relationship network and the current reality. As AI behavior tree frameworks, the commonly used lightweight behavior trees in Unity include Behavior Designer, NodeCanvas, and BTFlow, each with its own advantages and applicable scenarios. In this framework, BTFlow is used as the behavior tree editor. In the BTFlow editor, the various nodes and connections of the behavior tree are created, edited, and configured using a drag-and-drop approach. The BTFlow editor uses only three types of nodes: action nodes, condition nodes, and control nodes. Action nodes are responsible for executing specific behaviors, such as movement, attack, and animation

FIGURE 5 LitJSON serializes the battlefield

playback. Condition nodes are used to determine whether certain conditions are met, such as whether the character is within attack range or has been damaged. Control nodes are used to control the execution flow of the behavior tree. Composite nodes have been added to this framework, mainly used to manage a group of child nodes and can achieve complex behavior tree logic, such as selection nodes, sequence nodes, parallel nodes, repeat nodes and so on. These nodes can be combined as needed to achieve more complex behavior tree structures. Composite nodes are widely used in game development and can be used to implement AI behaviors, task systems, plot management and so on.

In addition, the editor in the framework provides a set of predefined nodes that represent common behaviors, such as moving, attacking, or searching for objects. It also allows the creation of custom nodes to achieve specific behaviors. The behavior tree system in this framework is designed to be easy to use and integrate into existing projects. It offers a simple API that allows developers to create and manage behavior trees in their code. Additionally, it supports saving and loading behavior trees from XML files. Overall, the behavior tree system in the framework is a powerful tool that can assist game developers in creating complex and dynamic AI behaviors for their games.

3.2.3 | Game flow tool

The Game Flow Tool is a state machine-based tool that assists game developers in quickly creating game flows. The tool manages the flow of the game by utilizing the state machine pattern. This involves defining states and transitions between them to implement the game logic. In the state machine, each state represents a stage or state of the game, and the transitions between them indicate the flow of the game. For example, in this framework, developers can create the game state by inheriting the "State" class and define the specific logic of the game state in this class. The transitions between states are implemented through the "Transition" class, which allows developers to specify the conditions for transitioning between states and the associated behavior. To compute the attack capability in the ESC architecture, developers can break it down into different components such as the base attack power, equipment attack power, percentage provided by equipment, and buff attack power. Attributes are divided into several categories, including the base value, equipment bonus, equipment percentage, buff bonus, and buff percentage. Developers can use these categories to compute the overall attack power of the player. Finally, the framework also includes LitJSON's built-in JsonMapper, which enables developers to serialize .NET objects to JSON strings or deserialize JSON strings to .NET objects. This makes it easier to manage and modify mod tables with complex data structures. These tools and libraries significantly reduce the time and effort required to interact with mod tables, allowing developers to focus more on game development rather than file management (Figure 6).

The framework used string extension methods to streamline the conversion process of document content into the properties needed for shaping, coloring, and other purposes: Signed shaping versus unsigned shaping. In Unity, signed

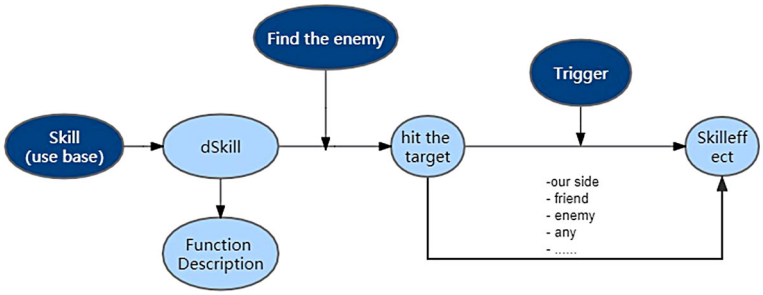


FIGURE 6 Skill system function flow

TABLE 1 Window type

Name	Remarks	Function
Window	Window	A general window, placed at the bottom.
Pop	Pop-up window	Above the window, generally used as a popover.
Loading	loading	Above the popup, generally used as a page load.
Max	highest	Only for special purposes, with the highest level.

and unsigned shaping are not directly relevant concepts. However, when working with low-level graphics programming in Unity, we may encounter signed and unsigned integer types. Boolean operations: They are used for making decisions based on logical conditions, controlling the flow of code, and implementing behaviors. The basic Boolean operations in Unity are the same as those in other programming languages: AND, OR, and NOT. Types of fractions: However, Unity does provide some support for working with fractions in the form of the Mathf class, which includes methods for manipulating rational numbers. These methods can be useful for performing calculations involving angles, distances, and other geometric quantities. Floating-point numbers: Floating-point numbers are the most commonly used numerical data type in Unity. They are used to represent values with a decimal component, such as positions, velocities, and angles. Color variations: Unity provides several tools and techniques for working with color variations, including the Color class and the Gradient class. The color class allows you to specify colors using RGB or HSL values, and to manipulate them using various methods. The gradient class allows you to create smooth transitions between colors, which can be used for things like creating color ramps or animating color changes over time. Vectors in two and three dimensions. Unity provides several built-in vector classes, including Vector2, Vector3, and Vector4, which can be used to represent vectors in two, three, and four dimensions, respectively. These classes provide various methods for manipulating vectors, such as adding, subtracting, multiplying, and dividing them, as well as calculating dot products, cross products, and other vector-related quantities.

3.2.4 | UIManager

The UIManager in this game framework is set automatically. In such cases, the UIManager may be created and managed automatically based on the game's profile or other settings. For example, some frameworks can create UIManager instances based on specific tags or game objects in a scene and automatically add UI elements to the UI root node at runtime. The main interface has been designed to display the player's life, magic, and shield values, along with a quick props bar and function buttons. When the corresponding button is pressed, the UIManager for the desired interface will open. Furthermore, players can access the backpack and skills pages by using the B or K shortcut keys, respectively. The backpack screen includes a simple attribute viewing area, a backpack section, an equipment bar, and a shortcut bar. Since the shortcut bar is linked to all other shortcut bars, changing one will affect all of them. Players can drag weapons and items to the shortcut bar, and equipment can be dragged to the equipment bar. The window type and its closing behavior are defined first. The actions for closing or destroying a window, as well as opening and activating a window, are defined below. There are four window types, as shown in Table 1.

In addition to the aforementioned structures, the TRPGFramework also includes hierarchical property actions, object pools, timer managers, event managers, and resource managers. The authors have designed the framework to be as lightweight as possible, simplifying its complexity so that developers can quickly understand and start using it. This framework can be applied to any type of game, not just RPGs. Despite having fewer features than other frameworks, it is still fast and easy to use.

4 | TESTS AND CONCLUSIONS

To validate the game framework, a 2D RPG game was used as an example. The main goal was to ensure the game is playable and that the system meets the specified requirements. It is well-known that an official 2D game project resource consists of pages, prefabs, scripts, audio, sprites, and other files. The project utilizes resource packs such as Ads, Analytics Library, Cinemachine, HD Render Pipeline, In-App Purchasing, Lightweight Render-Pipeline (LWRP), Shader Graph, and others. Certain resource packs such as Lightweight RenderPipeline and Shader Graph are required for this project, as they provide simplified, physics-based lighting and materials for LWRP, allowing the creation and editing of shaders by dragging and dropping in the graphical user interface. The official program structure resembles that of an MVC framework, with a game scene divided into four major components: the time management system, the user interface, player assets, and level assets. Smaller components such as HealthConvas and Footlight are placed below these four components. The design consists of modules such as ZGamework (Event events, UI windows, object pools), which uses UseBase to automatically populate the use method whenever an item in the prop bar, such as consumables, weapons, equipment, and skills, is used. It is very simple to call this method and be done with it. Normally, the UI can be opened and closed, and the data and props in the backpack are displayed, as shown in Figure 7. Effective data access and rendering optimization are utilized for smooth user interaction.

A good game development framework should not only complete the designed function, but also have good performance. In order to find the defects of this framework in time, improve its efficiency and improve its shortcomings, this article will carry out performance testing on the framework. In order to evaluate the performance of the TRPGFramework, we rebuilt the 2D RPG using this framework. We analyzed the event system test and logical code test, respectively. The skill system is composed of various parts, including directed attacks, damage, skill boosts and bonuses, undirected damage, automatic attacks, manual attacks, group attacks, and individual attacks, all of which are included in each skill module. The skill module has three levels, and the system creates a variety of interesting abilities. In the testing section, various types of skills are demonstrated in detail. Target selection is performed according to the skill's target selection method, and the target is fine-tuned and modulated based on specific parameters. The final target is locked, and EffectTo is called on it to add the skill effect. Figure 8 shows the effects of these four skills, which can be switched between using

FIGURE 7 UI display (ability to open and close menu, with data and items in the backpack displayed properly)

the middle mouse button. Along with the skills system, the TRPGFramework also includes UnitManager and BuffManager modules. Starting with Unity version 2020.3, we ran a benchmark on a 64-bit Windows 10 system with an Intel Core i9-10900k CPU and a 2080TI GPU.

4.1 | Event system test

Unity's SendMessage method is implemented through reflection. To send a message, the sender needs to get a reference from the subscriber. The difference between using SendMessage and calling methods directly on an object is that SendMessage can call private methods on the object. To test the performance of a game framework using SendMessage, we can follow these steps: First, create a test scenario that contains the game frame and related objects that we want to test. Then, create a TestScript script and write the test code by calling the function in the game frame with the SendMessage function and passing the parameters. After that, we can run the test scenario with Unity Test Runner and record the test results, such as execution time and memory usage. These results can help us evaluate the performance of the game framework and identify any potential areas for optimization.

We conducted a test to compare the performance of Unity's event system and its built-in SendMessage method. The code for this test is shown in Figure 9. The test involved sending 200 messages to each of 20 subscribers using the SendMessage method. To measure the CPU and memory usage of the events, we utilized Unity Engine Profiler tool to sample their performance. Unity's SendMessage method uses reflection to send a message. The sender needs to obtain a reference from the subscriber in order to send the message. The advantage of using SendMessage over calling methods directly on an object is that SendMessage can call private methods on the object.

Table 2 shows that the framework has twice the latency and memory consumption compared to Unity's built-in SendMessage method, and it does not perform as well when synchronizing messages. The reason behind this is that event systems based on the ECS framework are designed for asynchronous event flow, and packaging synchronous events as asynchronous events requires more time and memory. Asynchronous processing of the data stream reduces the latency and memory consumption during the send phase to a minimum. Although the latency and cost during the receive phase are higher than Unity's built-in SendMessage method, the excess is still within acceptable limits for the functionality improvement.

4.2 | Logical code test

By using the Profiler tool provided by the Unity engine, we sampled the CPU usage and memory usage of the sent events. Recording the events for the 100 rounds of the relay, we found that the traditional Unity Framework initialized

FIGURE 8 Four kinds of skills display image

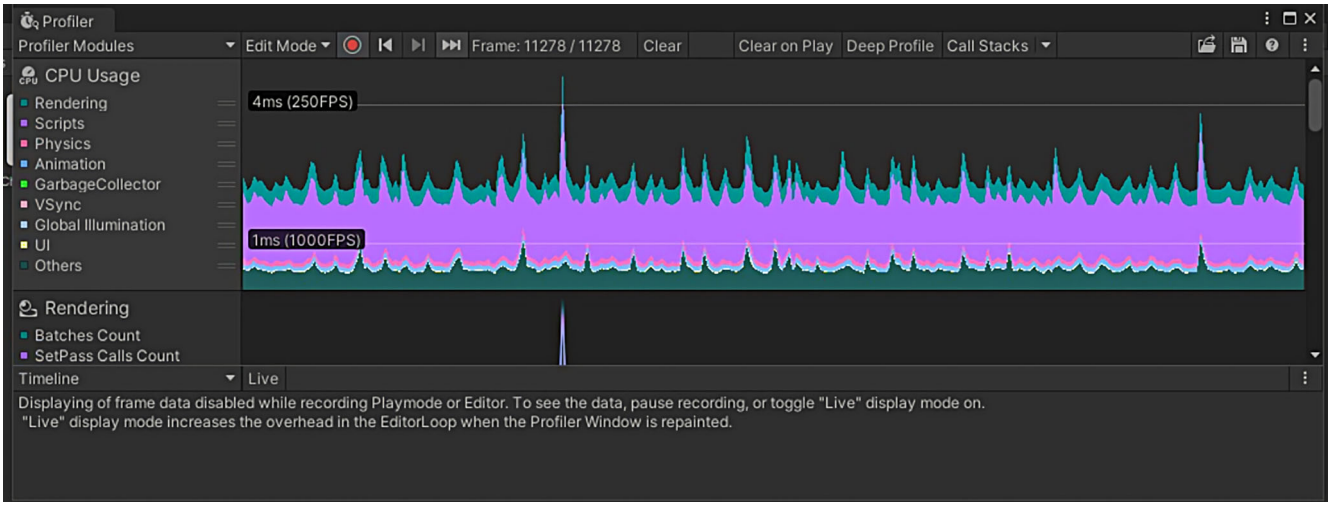


FIGURE 9 CPU usage in the framework

TABLE 2 Framework EventSystem test result

Event system type	Sync/async	Delay (ms)	Memory (MB)
Unity SendMessage	Sync	876.21	25.5
TRPGFramework	Sync	1652.32	39.7
TRPGFramework	Async	887.98	27.8

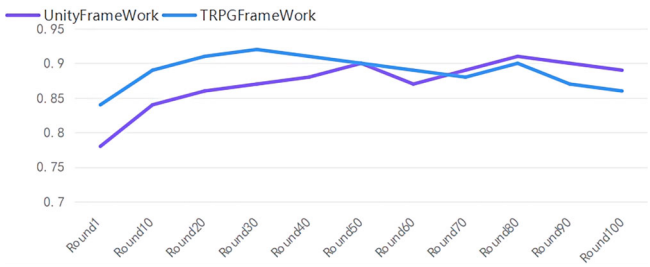


FIGURE 10 Test result line chart

faster than the TRPGFramework at the start of the game, and that the Unity Framework was faster during the run after initialization. However, the TRPGFramework ultimately won the 100 rounds of the relay. Using a sample of the data from the 100-round race, we plotted a line graph, as shown in Figure 10, to illustrate the performance gap between the two methods.

In the first round, the UnityFrameWork initializes 0.06 seconds faster than the TRPGFramework. The horizontal axis of Figure 10 displays the average time taken in the last 10 rounds, with the exception of the first round, which displays the exact time taken. Despite the fact that the logic developed in the ECS framework required a great deal of time in the first round, by the tenth round, it was taking roughly the same amount of time as the conventional approach. Over the next 80 rounds, both sets of logic take roughly the same amount of time on average, with the TRPGFrameWork taking slightly less time than the conventional development method in most cases, allowing the TRPGFrameWork-controlled runner to complete the 100-round relay first, 0.18 s ahead of the conventional development method. Although the TRPGFrameWork is slightly slower than the UnityFrameWork during initialization, the logic controlled by the TRPGFrameWork after relatively long runs is significantly more efficient.

The TRPGFramework is a comprehensive evaluation of the TRPGFramework's functionality, usability, reliability, performance, and supportability.

The practical application enables us to identify the framework's flaws and address them. We rewrote the official Unity demo to ensure comparability and practicality when implementing the TRPGFrameWork in practice. The TRPGFramework methods have significant functional advantages over traditional development methods, according to a performance comparison of the two development approaches. In previous performance tests, we evaluated the performance of the TRPGFrameWork using test cases. Now, we will compare the two in the context of an RPG game application. For this analysis, we will continue to use the Unity engine's profiler to obtain valid data. We have chosen the point in the game when the player has killed 1000 enemies for comparison to figure out how well the game works when played normally.

5 | DISCUSSION

The TRPGFramework in Unity is comprised of a lightweight gaming framework and several RPGs featuring in-game development and digital planning functionalities. However, it was specifically designed to address particular issues in certain RPGs and may not be suitable for all game types. Therefore, developers should be aware of its limitations before using it and ensure it is appropriately customized for optimal results. The TRPGFramework offers essential lightweight functionalities, including an event system, a timer manager, object pools, multi-level properties, and string expansion. The proposed architecture provides various benefits, such as flexibility, extensibility, and a clear separation of state and behavior. When combined with Odin and addressable tools, it results in a streamlined and widely applicable game framework. This RPG framework can even be used to create real-time combat RPGs, which are among the most complex game types available today. The buff system in this framework utilizes the most effective elements of the current high-quality combat framework, EGamePlay, making it easy to control and modify.

This framework is a combination of ideas from Kingdom Come 3, The Greatest Showman, and The Original Stone Project, which is a recent trend. It would be challenging for the average developer to create their own set of standalone mod designers or a complete set of fully open parameters, as Kingdom Come 3's development team did or the authors of The Original Stone Project did. Therefore, we chose the simplest and most effective way to use it by storing data directly in JSON and XML, enabling developers to decide for themselves whether to allow player customization. In future game development projects, the game framework will continue to be updated to provide more useful features while meeting the project's specific requirements.

FUNDING INFORMATION

Supported by Jinling Institute of Science Technology Research Initiation Project (jit-b-202030).

CONFLICT OF INTEREST STATEMENT

All authors disclosed no relevant relationships.

DATA AVAILABILITY STATEMENT

The data that support the findings of this study are available on request from the corresponding author upon reasonable request.

ORCID

Bo Zhang https://orcid.org/0000-0001-6933-4402

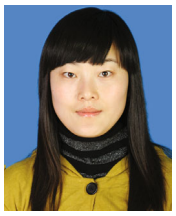
REFERENCES

1. Anderson C. The model-view-ViewModel (MVVM) design pattern. Pro Business Applications with Silverlight 5 the Model-View-Viewmodel (Mvvm) Design Pattern. Berkeley: Apress; 2012. p. 461–99. https://doi.org/10.1007/978-1-4302-3501-9_13
2. Gu MX, Tang K. Comparative analysis of webforms mvc and mvp architecture. Proceedings of the 2nd Conference on Environmental Science and Information Application Technology, Wuhan, China, pp. 391-394. IEEE, New York (2010).
3. Gamma E. Design patterns: elements of reusable object-oriented software (for jntu). A web-based application for real-time GIS. XXth ISPRS Congress, vol. 49. pp. 241-276 (1998).
4. Maleki S, Fu C, Banotra A, Zong Z. Understanding the impact of object oriented programming and design patterns on energy efficiency. Proceedings of rhe 8th International Green Sustainable Computing Conference, pp. 1-6. IEEE, New York (2017).

5. Hatledal LI, Chu Y, Styve A, Zhang H. Vico: an entity-component-system based co-simulation framework. Simul Model Pract Theory. 2021;108:102243.
6. Romeo V. Analysis of entity encoding techniques, design and implementation of a multithreaded compile-time entity-component-system c++14 library (2016).
7. Fischbach M, Wiebusch D, Latoschik ME. Semantic entity-component state management techniques to enhance software quality for multimodal VR-systems. IEEE Trans Vis Comput Graph. 2017;PP(4):1–1351.
8. Elvezio C, Sukan M, Feiner S. Mercury: a messaging framework for modular ui components. Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems, pp. 1-12. ACM (2018).
9. Ahmed S, Maunder RG, Yang LL, Ng SX, Hanzo L. Joint source coding, unity rate precoding and ffh-mfsk modulation using iteratively decoded irregular variable length coding. Proceedings of the IEEE 66th Vehicular Technology Conference. IEEE, New York (2007).
10. Zhao S, Xu Y, Luo Z, Tao J, Pan G. Player behavior model-ing for enhancing role-playing game engagement. IEEE Trans Comput Soc Syst. 2021;8:464–74.
11. Wang RX, Wang R, Fu P, Zhang JM. Portable interactive visualization of large-scale simulations in geotechnical engineering using unity3d. Adv Eng Softw. 2020;148(2020):102838.
12. Cao XR, Cohen G, Giua A, Wonham WM, van Schuppen JH. Unity in diversity, diversity in unity: retrospective and prospective views on control of discrete event systems. Discrete Event Dynamic Syst. 2002;12:253–64.
13. Lange P, Weller R, Zachmann G. Wait-free hash maps in the entity-component-system pattern for realtime interactive systems. Proceedings of the IEEE 9th Workshop on Software Engineering and Architectures for Realtime Interactive Systems (SEARIS), Greenville, SC; pp. 1-8. IEEE (2016).
14. Thule C, Lausdahl K, Gomes C, Meisl G, Larsen PG. CasperLausdahl: maestro: the into-cps co-simulation framework. Simul Model Pract Theory. 2019;92:45–61.
15. Callum KM, Jamieson J. Exploring armobile in early childhood literacy learning. In: Flexible Learning Association of New Zealand (FLANZ 2018) (2018).
16. Cieza E, Lujan D. Educational mobile application of augmented reality based on markers to improve the learning of vowel usage and numbers for children of a kindergarten in Trujillo. Procedia Comput Sci. 2018;130:352–8.
17. Raffaillac T, Huot S. Polyphony: programming interfaces and interactions with the entity-component-system model. Proc ACM Hum Comput Interact. 2019;3:1–22.
18. Muhammad K, Khan N, Lee M-Y, Imran AS, Sajjad M. School of the future: a comprehensive study on the effectiveness of augmented reality as a tool for primary school children's education. Appl Sci. 2021;11(11):5277.
19. Amin D, Govilkar S. Comparative study of augmented reality sdks. Int J Comput Sci Appl. 2015;5(1):11–26.
20. Harris S. Implementation and analysis of the entity (2022).
21. Bhattacharya I, Getoor L. Collective entity resolution in relational data. ACM Trans Knowl Discov Data. 2007;1(1):5.
22. Wu C-H, Wu D-Y, Chou H-M, Cheng C-A. Rethink the design of flash translation layers in a component-based view. IEEE Access. 2017;5:12895–912.
23. Sadjina S, Kyllingstad LT, Rindarøy M, Skjong S, Æsøy V, Pedersen E. Distributed co-simulation of maritime systems and operations. J Offshore Mech Arct Eng. 2019;141(1):011302.
24. Hatledal LI, Skulstad R, Li G, Styve A, Zhang H. Co-simulation as a fundamental technology for twin ships. Model Identif Control. 2020;41:297–311.
25. Basile D, ter Beek MH, Ciancia V. An experimental toolchain for strategy synthesis with spatial properties. Leveraging applications of formal methods, verification and validation. Adaptation and learning: 11th international symposium, ISoLA 2022. Rhodes, Greece: Springer; 2022. p. 142–64.
26. Jadon S, Singhal A, Dawn S. Military simulator-a case study of behaviour tree and unity based architecture. arXiv:1405:7944 (2014).

AUTHOR BIOGRAPHIES

Bo Zhang received his Ph.D. degree in Computer Application Technology from Jilin University in 2018. People's Republic of China. From 2009-Now, he worked as teather in the College of Computer Engineering, the Jinling Institute of Technology University, Nanjing, People's Republic of China. His current research focuses on unity multimedia applications, game framework building and related subject areas, including elemental human production, game frameworks and game development. He has published more than 10 papers in many peer-reviewed journals over the past 10 years. He has published more than 15 articles in core journals in China.



Huiping Shi was born in Jiaohe, Jilin, People's Republic of China, in 1984. She received the master's degree from Liaoning Normal University, People's Republic of China. Now she works as a Japanese teacher at the School of Foreign Languages, Jinling Institute of Technology. Her research interests include applied linguistics and research on Japanese teaching methods.

Xinyu Wang in 2022, he earned a Bachelor's degree in Digital Media Technology from Jinling Institute of Technology. His ongoing research centers around VR applications, game framework development, and related fields, including interaction design and game development, among others.

How to cite this article: Zhang B, Shi H, Wang X. An auxiliary development framework for lightweight RPG games based on Unity3D. Comput Anim Virtual Worlds. 2024;35(1):e2206. https://doi.org/10.1002/cav.2206