FRI-1.407-1-MIP-03

THE ROLE OF SCRIPTING IN THE GAME ENGINE

Mustafa Mustafov – Student
Faculty of Natural Sciences and Education
University of Ruse "Angel Kanchev"
Tel.: +359890569310
E-mail: s216259@uni-ruse.bg

Assoc. Prof. Kamelia Shoilekova, PhD
Department of Informatics and Information technologies
Faculty of Natural Sciences and Education
University of Ruse "Angel Kanchev"
Tel: +359887859224
E-mail: kshoylekova@uni-ruse.bg

Abstract: *A game engine is a software development environment with settings and configurations that optimize and simplify the creation of video games using various programming languages — faster game development results from reusing the built systems within the game engine for different purposes.*

This publication aims to show that scripting is the bridge between the programmer and the built systems, providing access and control over the logic. Through scripting, modules are created that, as a result of reflection, offer a view of all variables and functions, allowing their visualization in the application's interface. This facilitates the use of the developed reflection module by others. The integration of scripting into the game engine involves creating a secondary project where the logic for modules is written, which is compiled into a dynamic library. The game engine loads this library at runtime, extracts all the reflection information (from the secondary project), and feeds the data into the already established systems of the game engine.

Key words: *Scripting, Game Engine, Reflection, Software Development.*

INTRODUCTION

In the modern video game industry, game engines play a crucial role in creating interactive and engaging experiences for users. One of the most important aspects of game engines is their scripting capability, which allows developers to implement dynamic behavior and customize their games according to user needs. This paper explores the importance of scripting in game engines, focusing on the integration of dynamic self-analysis systems and the creation of scripting modules written in C++.

Dynamic self-analysis provides flexibility in the manipulation of game objects and mechanics and allows developers to interact with data in an intuitive way. By examining the architecture of this system and its impact on the performance and flexibility of game applications, it becomes clear that effective scripting not only speeds up the development process, but also enhances the creative potential of games. This paper will illustrate how the proper implementation of scripting mechanisms can improve the quality of the gaming experience through fast data processing, a benefit of C++'s high performance.

EXPOSITION

The Role of Scripting in Game Engines

The primary objective of this research is to investigate the role of scripting in game engines through the development of a custom system for dynamic self-analysis and scripting implemented in C++. This system offers a high-performance mechanism for manipulating game objects and dynamic behavior while providing developers with flexibility in interacting with game logic [1].

Central to this system is the concept of dynamic self-analysis, which enables game objects to be described and manipulated during runtime. This approach significantly streamlines the workflow by eliminating the need for manual data structure management and facilitating automated linking of scripting modules with game objects. Consequently, this methodology accelerates development, enabling rapid prototyping — a critical requirement in the gaming industry.

The utilization of C++ as the primary language for implementing these mechanisms confers considerable performance advantages. Its capacity to operate at a low level with memory and system resources ensures more efficient data processing compared to alternative scripting languages, resulting in enhanced engine efficiency, particularly in complex scenes with numerous game objects and interactive elements [2], [4].

Specific instances of applying dynamic self-analysis and scripting in real game scenarios will be presented subsequently, demonstrating the advantages of this approach.

Dynamic Self-Analysis (Reflection)

Dynamic self-analysis in programming is a technique whereby a program can examine its own structure, extracting information about all fields and allowing their values to be modified at runtime. This is particularly advantageous in the context of game engines, where developers can utilize this information to automate tasks such as managing game objects, serializing data, dynamically linking scripts, and creating intuitive development tools.

It is noteworthy that while the concept of dynamic self-analysis is highly beneficial, official support for it in C++ is still in the planning stages, anticipated in the C++26 standard. At present, C++ does not offer built-in mechanisms for full dynamic self-analysis, necessitating developers to rely on custom solutions for this functionality. These custom reflection systems are typically created manually and often involve registering data types and class members through macros or metaprogramming to enable dynamic data handling at runtime [3].

This paper examines such a custom solution for dynamic self-analysis, which facilitates easy access to and manipulation of game objects and components.

Scripting

Scripting involves the use of high-level languages that allow developers to quickly and easily add and modify game logic without needing to recompile the entire game engine. This is a crucial aspect of modern video game development, as it enables developers to implement new features and adapt flexibly to design changes and project requirements.

In scripting, core (or internal) logic is separated from game-specific logic, achieving system abstraction that streamlines workflow and reduces the risk of errors. Additionally, modules should be intuitively designed to be accessible to non-programmers, such as designers or testers, allowing them to interact directly with game mechanics.

Dynamic Self-Analysis Library

The development of a library for dynamic self-analysis presents significant challenges that necessitate extensive research and a comprehensive understanding of reflection and metaprogramming mechanisms. The fundamental aspect of this system involves the registration of data types and class members. The library utilizes macros to facilitate the registration process, thereby providing developers with an efficient method for incorporating reflection into their projects.

The dynamic self-analysis and scripting system offers functionalities for retrieving data types. For instance, the type of a variable can be obtained by invoking TypeOf (myVariable), which returns the type information for that variable – if the variable's type is "int", it will return "int". Furthermore, the library enables type retrieval by name using TypeOf ("bool"), offering additional flexibility when working with dynamically defined types.

The dynamic self-analysis and scripting system introduces three primary macros: TClass, TFunc, and TProperty.

- TClass: Utilized to define a class for reflection, capturing its name and properties.
- TFunc: Registers functions within a class, enabling access and dynamic invocation of methods at runtime.
- TProperty: Facilitates the registration of class properties, providing straightforward access to their data.

```
TClass()
class BoxLogic : public Entity
{
public:

    TFunc()
    void Start();

    TFunc()
    void Update(float dt);


    TProperty()
    glm::vec2 m_TargetLocation;
};
```

Fig. 1. Example interface of a module

Figure 1 illustrates sample code for a C++ script class designated as BoxLogic. This code employs macros such as TFunc() and TProperty(), which are essential components of the class structure.

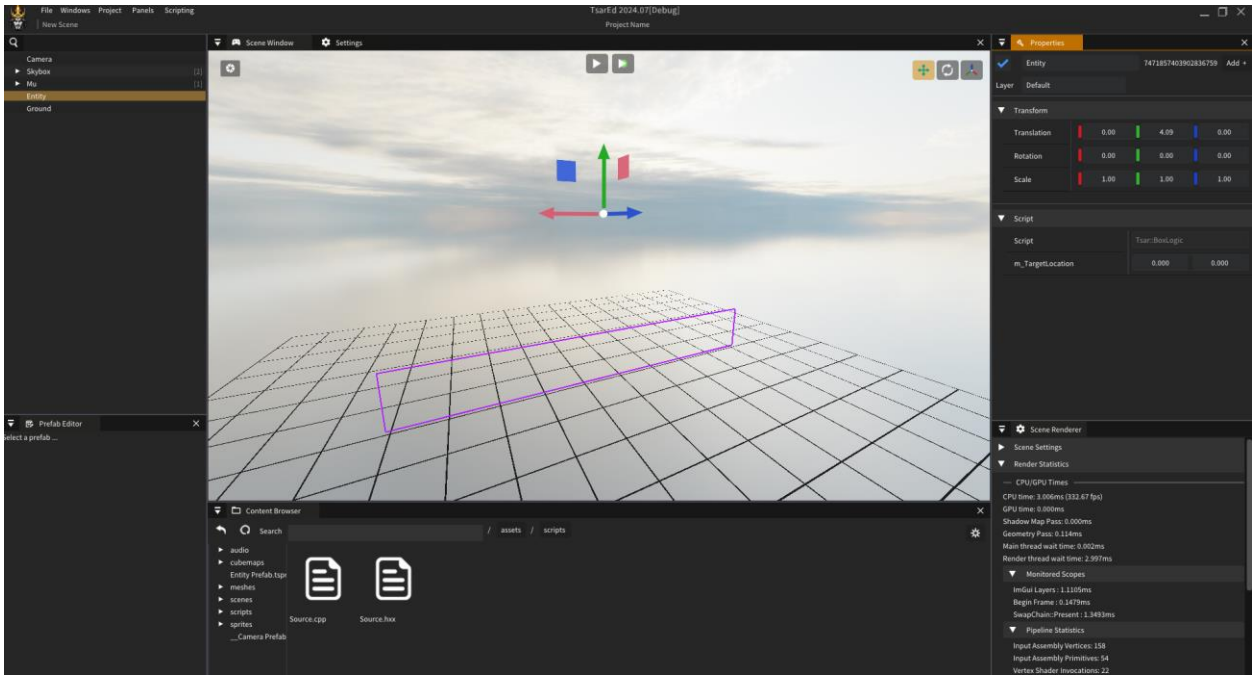

Fig. 2. Demonstration of the Game Engine in the Game Engine Interface

A game engine is a platform that provides tools for creating interactive and visual environments, with applications in video games, architectural visualization, training simulators,

and the film industry. Scripting affords game developers a flexible environment for manipulating and rapidly prototyping game mechanics, enabling them to create and modify object behaviors in real-time. Scripts simplify the process of adding new features and automating tasks, thereby enhancing the dynamism and adaptability of gameplay to project requirements.

Figure 2 illustrates the game engine interface, demonstrating the modular structure of scripting. The right side of the interface displays the "Properties" panel, where developers can configure the selected module's properties.

1. Script and Parameters: In the Script section of Figure 2, the attached module, in this case, "BoxLogic", is presented. Below it are the module's parameters, which can be modified in real time, enabling developers to adjust the game object's behavior without altering the engine's core code.
2. Modular Reusability: Figure 2 emphasizes that each script is a module that can be reused in various contexts. This is particularly advantageous for optimizing workflow, as the same script can be attached to multiple game objects.
3. Interactivity: The "Properties" panel in Figure 2 demonstrates the dynamic nature of the interface, where any parameter changes in the script immediately affect the behavior of the game object in the scene, providing developers with real-time feedback.

CONCLUSION

This study investigates the role of scripting in game engines and presents our custom library for dynamic self-analysis in C++. The development of this library has contributed to the successful implementation of a robust mechanism for dynamic data management, enabling developers to create and modify objects in a flexible and efficient manner.

By providing functions such as TypeOf ("int") and TypeOf ("bool"), the library facilitates data type information retrieval and property management. Furthermore, the Type class supports handling parent and child classes, as well as methods for manipulating field values.

Throughout the game engine's development, we emphasized a modular structure for scripts, facilitating their use and reintegration across different game contexts. This approach offers substantial flexibility and efficiency in game development, equipping developers with tools for implementing complex game mechanics.

In future work, we intend to continue optimizing and expanding the dynamic self-analysis library and incorporate new features and improvements to the game engine to address the evolving needs of modern game developers.

REFERENCES

1. Kupiainen, Hamza. Extending unity game engine through editor scripting. 2018.

2. Madina, Duraid; Standish, Russell K. A system for reflection in C++. Proceedings of AUUG2001: Always on and Everywhere, 2001, 207.

3. MÁRTON, Gábor; PORKOLÁB, Zoltán. C++ compile-time reflection and mock objects. Studia Univ. Babes–Bolyai Informatica, 2014, 59.2.

4. Sobel, Jonathan M.; Friedman, Daniel P. An introduction to reflection-oriented programming. In: Proceedings of reflection. 1996. p. 263-288.

