УДК 004.415.23

**І. Р. Арсенюк**
**П. М. Столяр**

# АРХІТЕКТУРА ПРОГРАМНОГО МОДУЛЯ ЗБЕРЕЖЕННЯ ІГРОВОЇ СИТУАЦІЇ В UNITY

Вінницький національний технічний університет

***Анотація***

*У роботі представлено результати проєктування програмного модуля збереження ігрової ситуації для рушія Unity. Обґрунтовано перехід від рефлексивного пошуку об'єктів, що зберігаються через FindObjectsOfType до самореєстрації компонентів у глобальному реєстрі на основі HashSet. Архітектуру побудовано за моделлю «оркестратор + незалежні сервіси»: центральний клас керує послідовністю викликів, а підсистеми серіалізації, стиснення, шифрування, версійної міграції та відкату стану залишаються відокремленими одна від одної. Бінарну серіалізацію виконано засобами MessagePack-CSharp у режимі IntKey з блочним стисненням LZ4. Конфіденційність і цілісність файлу сейву забезпечено схемою «Encrypt-then-MAC» на основі AES-256-CBC та HMAC-SHA256, причому вибір саме CBC замість GCM продиктовано вимогою сумісності з компілятором IL2CPP. Експериментальні дослідження підтверджують працездатність модуля для типових ігрових сценаріїв.*

***Ключові слова:*** *збереження ігрового стану, рушій Unity, бінарна серіалізація, MessagePack, LZ4, AES-256-CBC, HMAC-SHA256, відкат стану, версійна міграція, Roslyn Source Generators.*

**Abstract**

This paper presents the results of designing a software module for saving game state in the Unity engine. The transition from reflective object discovery via FindObjectsOfType to self-registration of components in a global HashSet-based registry is substantiated. The architecture follows the «orchestrator + independent services» model: a central class manages the call sequence, while serialization, compression, encryption, version migration, and rollback subsystems remain decoupled. Binary serialization is implemented using MessagePack-CSharp in IntKey mode with LZ4 block compression. Confidentiality and integrity of the save file are ensured by the «Encrypt-then-MAC» scheme based on AES-256-CBC and HMAC-SHA256, with CBC chosen over GCM due to IL2CPP compatibility constraints. Experimental research confirms the module's operability in typical game scenarios.

***Keywords:*** *game state saving, Unity engine, binary serialization, MessagePack, LZ4, AES-256-CBC, HMAC-SHA256, state rollback, version migration, Roslyn Source Generators.*

## Вступ

Сучасні відеоігри оперують десятками і сотнями динамічних об'єктів сцени, кожен з яких має власний стан. Збереження прогресу з часом перестало бути задачею запису кількох числових параметрів і перетворилося на серіалізацію складного контексту з вимогами до швидкості, обсягу, цілісності та можливості еволюції формату між версіями гри [1]. Вбудовані засоби Unity (PlayerPrefs, JsonUtility) орієнтовані на роботу з простими типами даних і не передбачають ні криптографічного захисту, ні версійної міграції, ні безпечного відкату до попередніх станів.

Поширені сторонні рішення частково знімають ці обмеження, проте успадковують характерні архітектурні недоліки: пошук об'єктів через сканування сцени, відсутність перевірки цілісності зашифрованих даних, брак механізму повернення до попереднього збереження [2]. Жодне з

розглянутих рішень не поєднує бінарну серіалізацію, автентифіковане шифрування та інструменти відмовостійкості в межах єдиної архітектури.

З огляду на це у роботі пропонується спроєктувати програмний модуль, у якому всі названі механізми працюють разом. Окремою задачею при цьому є також забезпечення стабільної роботи в умовах AOT-компіляції (Ahead-Of-Time compilation) на мобільних платформах, де рантайм-рефлексія недопустима.

**Результати дослідження**

Архітектуру модуля сформовано за моделлю «оркестратор + незалежні сервіси». Центральний клас SaveManager керує лінійною послідовністю операцій збереження та завантаження і не втручається у внутрішню реалізацію жодної з підсистем. Сервіси серіалізації, шифрування, стиснення, фонового автозбереження, версійної міграції та відкату взаємодіють між собою винятково через цей оркестратор. Реєстр об’єктів, що зберігаються реалізовано як статичну колекцію HashSet, до якої компоненти додаються в OnEnable і видаляються в OnDisable; такий підхід усуває потребу у викликах FindObjectsOfType, обчислювально витратних на середніх і великих сценах [3]. Загальну організацію модуля, центральний оркестратор та підпорядковані йому сервіси наведено на рис. 1.

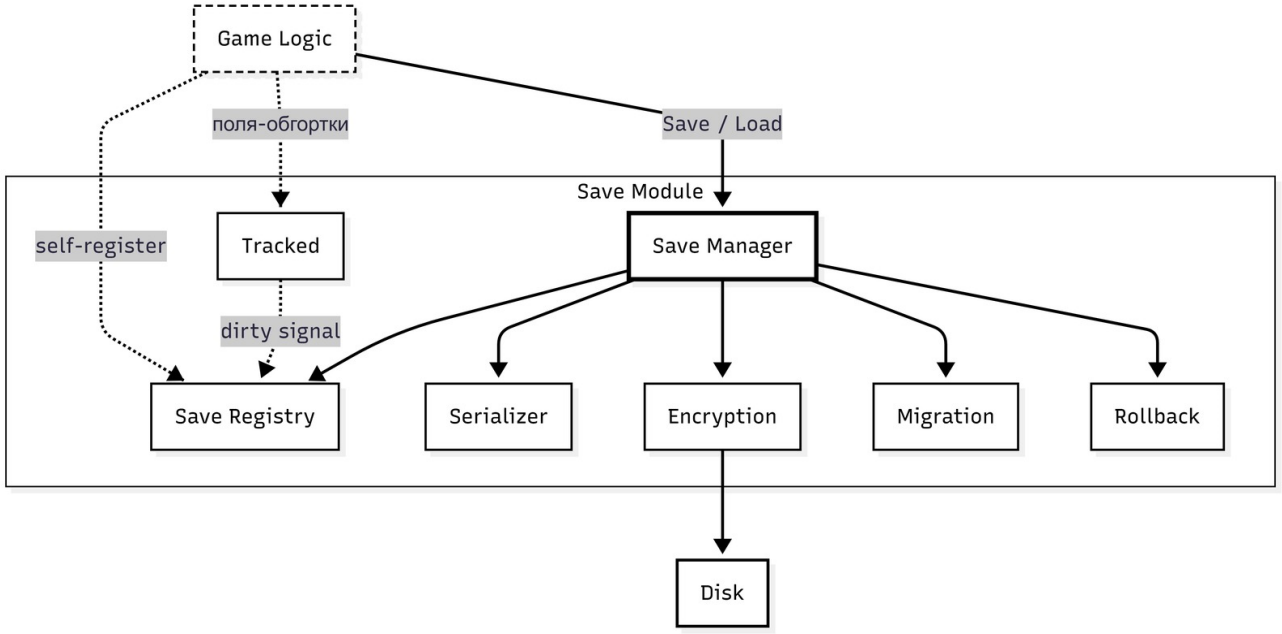


Рисунок 1 – Архітектура програмного модуля збереження ігрової ситуації

Бінарну серіалізацію виконано засобами бібліотеки MessagePack-CSharp у режимі IntKey (цілочислових ключів) з вбудованим блочним стисненням Lz4BlockArray [4]. MessagePack – це бінарний формат обміну даними, що його автори описують як «JSON (JavaScript Object Notation), тільки швидкий і компактний»: він зберігає ту саму модель «ключ–значення», проте не у вигляді тексту, а у вигляді щільно упакованих байтів; конкретну реалізацію від компанії Cysharp обрано через те, що вона помітно оптимізованіша за оригінальну реалізацію MsgPack-Cli – типізовані класи-перетворювачі для кожного типу створюються ще на етапі компіляції, без рантайм-рефлексії (Runtime Reflection – аналізу структури типів під час виконання), а сама десеріалізація не породжує зайвих алокацій пам'яті, що для Unity важливо, бо провокування збирача сміття (Garbage Collector, GC) посеред кадру дає помітні просідання частоти кадрів. Режим IntKey означає, що поля класу позначаються не рядковими іменами, а цілочисловими індексами через атрибут [Key(0)], [Key(1)] і далі, тож у файл потрапляє не назва поля, а лише його номер – представлення виходить компактнішим і трохи швидшим у читанні, ніж у режимі зі строковими ключами (StringKey); платою за це є

дисципліна додавання нових полів виключно в кінець класу, інакше індекси зсунуться і старі сейви прочитаються неправильно. Стиснення покладено на алгоритм LZ4 (Lempel–Ziv, четверта версія) з акцентом саме на швидкість розпакування, а не на максимальний коефіцієнт – його декодер працює майже на швидкості читання з оперативної пам'яті, що для сейвів доречно, оскільки завантаження виконується частіше за збереження; варіант Lz4BlockArray (блочний масив LZ4) додатково розбиває дані на окремі блоки і стискає кожен незалежно, дещо прискорюючи розпакування великих файлів. Обидва кроки – серіалізація і стиснення – викликаються одним методом і працюють як єдиний конвеєр, без проміжного копіювання буферів; для тестового набору з десяти компонентів розмір записуваного файлу (сейву) зменшується з 1309 байт (JSON) до приблизно 600 байт (MessagePack + LZ4), і основну економію дає саме перехід до бінарного формату, тоді як додаткове стиснення скорочує об'єм приблизно вдвічі.

Захист цілісності та конфіденційності файлу збереження побудовано за схемою «Encrypt-then-MAC» (спочатку шифрування, потім обчислення коду автентифікації повідомлення; MAC – Message Authentication Code) [5, 6]. Дані шифруються алгоритмом AES-256-CBC (AES – Advanced Encryption Standard, симетричний блоковий шифр із 256-бітним ключем; CBC – Cipher Block Chaining, режим зчеплення блоків шифротексту) з випадковим 128-бітним вектором ініціалізації (IV, Initialization Vector), а над зашифрованим масивом обчислюється підпис HMAC-SHA256 (HMAC – Hash-based Message Authentication Code, код автентифікації на основі хеш-функції; SHA-256 – Secure Hash Algorithm з виходом 256 біт). На етапі завантаження порядок зворотний: спочатку звіряється підпис, і лише за умови збігу виконується дешифрування; саме порівняння реалізовано за сталий час, щоб ускладнити можливі таймінг-атаки. Вибір саме CBC замість сучаснішого GCM (Galois/Counter Mode – режим лічильника Галуа) зумовлений практичними міркуваннями. Клас AesGcm у Unity при компіляції через IL2CPP (транслятор проміжного коду в C++) на мобільних платформах часто спричиняє виключення PlatformNotSupportedException, тоді як CBC підтримується стабільно. Криптографічні накладні витрати (оверхед) залишаються фіксованими – близько 48 байтів незалежно від розміру сейву.

Для уникнення рантайм-рефлексії в AOT-збірках на iOS і консолях розроблено генератор коду на основі технології Roslyn Source Generators, який на етапі компіляції створює часткову (партіальну) реалізацію інтерфейсу ISaveable для класів з атрибутом [Saveable]. Підсистему відкату реалізовано через кільцеву чергу незмінних знімків (за замовчуванням – п’ять останніх вдалих збережень), а версійну міграцію – як ланцюжок незалежних перетворень, де кожен мігратор знає лише про перехід N+1. Порівняльні характеристики розробленого програмного модуля та стандартних засобів Unity наведено нижче у табл. 1.

Таблиця 1 – Порівняльний аналіз підходів до збереження ігрового стану в Unity

| Критерій порівняння | Стандартні засоби Unity | Розроблений програмний модуль |
|---|---|---|
| Виявлення об’єктів, що зберігаються | Сканування сцени | Самореєстрація в HashSet |
| Формат серіалізації | JSON або примітивні типи | Бінарний (MessagePack + LZ4) |
| Захист цілісності | Відсутній | HMAC-SHA256, перевірка перед дешифруванням |
| Підтримка версій | Ручна | Ланцюжок мігруючих перетворень |
| Відкат до попередніх станів | Не передбачено | Кільцева черга з п’яти знімків |
| Сумісність з AOT/IL2CPP | Часткова | Повна (через Source Generators) |

## Висновки

Розроблено архітектуру та програмний модуль збереження ігрової ситуації в Unity, де поєднано бінарну серіалізацію MessagePack зі стисненням LZ4, схему «Encrypt-then-MAC» на основі AES-256-CBC та HMAC-SHA256, кільцеву чергу відкату й ланцюгову версійну міграцію. Самореєстрація об'єктів у HashSet та автоматичне формування коду за допомогою Roslyn Source Generators дають змогу уникнути типових труднощів, пов'язаних із рефлексією типів під час виконання в умовах IL2CPP. Експериментальні дослідження підтверджують працездатність обраних рішень у типових випадках, але водночас показують, що окремі оптимізації виправдані лише за певних умов, а не за замовчуванням.

У подальшому пропонується даний програмний модуль модифікувати у напрямі асинхронного запису та інтеграції з хмарними сервісами синхронізації.

## СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ


1. Як з’явилися збереження в іграх: від «темних часів» до хмарних сервісів [Електронний ресурс] // ITC.ua. – 2025. – Режим доступу: https://itc.ua/ua/blogs/yak-z-yavylysya-zberezhennya-v-igrah/
2. Easy Save – The Complete Save Data & Serialization Asset [Електронний ресурс] // Unity Asset Store. – 2026. – Режим доступу: https://assetstore.unity.com/packages/tools/input-management/easy-save-the-complete-save-data-serialization-asset-768
3. Unity Documentation: Object.FindObjectsOfType [Електронний ресурс]. – Режим доступу: https://docs.unity3d.com/ScriptReference/Object.FindObjectsOfType.html
4. MessagePack-CSharp [Електронний ресурс] // GitHub. – Режим доступу: https://github.com/MessagePack-CSharp/MessagePack-CSharp
5. Bellare M., Namprempre C. Authenticated Encryption: Relations among notions and analysis of the generic composition paradigm // Journal of Cryptology. – 2007. – Режим доступу: https://eprint.iacr.org/2000/025.pdf
6. Самойлов І. В., Матійко А. А., Сторчак А. С. Криптографія [Електронний ресурс] : навч. посіб. / КПІ ім. Ігоря Сікорського. – Київ, 2023. – 372 с. – Режим доступу: https://ela.kpi.ua/server/api/core/bitstreams/5faf3d32-f858-4eec-8f10-b059dc28d7e4/content



***Арсенюк Ігор Ростиславович*** *– канд. техн. наук, доцент кафедри комп’ютерних наук, Вінницький національний технічний університет, м. Вінниця, e-mail: air@vntu.edu.ua*

***Столяр Павло Михайлович*** *– студент групи 5КН-22б, факультету інтелектуальних інформаційних технологій та автоматизації, Вінницький національний технічний університет, м. Вінниця, e-mail: kilvaprylost@gmail.com*

***Arseniuk Igor R.*** *– Associate Professor of the Computer Sciences Department, Vinnytsia National Technical University, Vinnytsia, e-mail: air@vntu.edu.ua*

***Stolyar Pavlo M.*** *– Student of the 5CS-22b group, Department of Intellectual Information Technologies and Automation, Vinnytsia National Technical University, Vinnytsia, e-mail: kilvaprylost@gmail.com*