УДК 004.4:004.75

СПЕЦИФІКА КОНТРОЛЮ ВЕРСІЙ У РОЗРОБЦІ ВІДЕОІГОР

Богуцька А.О.
Національний університет «Одеська юридична академія»
Україна, Одеса
anastasiyabogutskaya09@gmail.com

***Анотація**. Проаналізовано специфіку використання систем контролю версій у розробці відеоігор з урахуванням поєднання програмного коду та різнорідних цифрових ресурсів: 3D-моделей, текстур, аудіо, анімацій і файлів сцен. Розглянуто проблеми версіонування великих бінарних об'єктів, обмеженість автоматичного злиття та роль блокування файлів у командній роботі. Виявлено вимоги геймдеву до історії змін і спільного редагування.*

***Ключові слова**: контроль версій; розробка відеоігор; бінарні ресурси; спільне редагування; блокування файлів; Git LFS; Perforce.*

Вступна частина. Розробку відеоігор відрізняє від інших напрямів програмної інженерії те, що результат створюють спільно фахівці із програмування, дизайну, 3D-графіки, анімації, звуку та проєктування рівнів. У межах одного проєкту співіснують текстові файли вихідного коду й конфігурацій, а також великі бінарні ресурси, зміни яких не завжди можна наочно порівняти або автоматично об'єднати. Тому система контролю версій (СКВ) у геймдеві виконує не лише функцію збереження історії коду, а й підтримує узгодженість цифрових ресурсів та командної роботи [1].

Мета роботи – аналітично дослідити специфіку контролю версій у розробці відеоігор, визначити особливості версіонування бінарних ресурсів, координації паралельної роботи та збереження історії змін.

Основна частина. На відміну від проєктів із текстовим кодом, ігрові проєкти містять різнорідні файли: моделі, текстури, звук, анімації, карти та сцени. СКВ порівнюють текстові файли порядково й часто зливають автоматично, а для багатьох бінарних форматів такий механізм недоступний. Наприклад, в Unreal Engine ресурси .uasset та .umap є бінарними і не можуть бути об'єднані стандартним текстовим засобом злиття [2]. Відтак конфлікт у такому файлі часто потребує координації авторів і вибору узгодженої версії, а не автоматичного злиття.

Контроль версій дає змогу фіксувати стан проєкту в часі, визначати авторство змін, повертатися до попередньої версії та підтримувати паралельну розробку різних функцій. У геймдеві ці можливості мають охоплювати не тільки код, а й контент, від якого залежить відтворюваність ігрового рівня чи поведінки об'єктів. Тому історія версій є не лише засобом відновлення файлів, а й джерелом інформації про організацію процесу розробки [3].

Коли два учасники одночасно редагують один і той самий ресурс, який не підтримує змістовного автоматичного злиття, виникає ризик того, що одна версія замінить іншу або доведеться вручну відтворювати частину роботи. Один із підходів – ексклюзивне блокування: за документацією Unreal Engine, редактор блокує файл на час редагування (check out) і знімає блокування після надсилання змін до сховища (check in) [2], а статус ресурсу, зокрема факт його редагування іншим користувачем, видно в редакторі [1]. Блокування зменшує ймовірність конфліктів, але не замінює збереження результатів, контролю доступу та резервного копіювання. Також воно не є обов'язковим для всіх типів файлів: доцільність визначається тим, чи можна безпечно об'єднати паралельні зміни.

Окрема проблема полягає в обсязі бінарних ресурсів та зростанні історії сховища. Git

Large File Storage (Git LFS) розв'язує її шляхом зберігання в Git текстового вказівника на великий об'єкт, тоді як сам вміст розміщується в окремому LFS-сховищі [4]. Така схема зменшує обсяг великих об'єктів у звичайній історії Git, однак потребує доступності сервера LFS і належної конфігурації завантаження ресурсів. Git LFS також використовує механізм блокування, але він залежить від підтримки відповідного API сервером і налаштувань клієнтів [5]. Тобто використання Git LFS не означає автоматичної тотожності робочого процесу централізованим системам: організація сховища, доступ до великих файлів і правила блокування мають бути налаштовані окремо.

Інтеграція системи контролю версій із редактором рушія є важливою складовою повсякденного робочого процесу. Вона дає змогу переглядати статус ресурсів, отримувати актуальні версії, реєструвати зміни та відстежувати файли, які редагують інші учасники. Документація Unreal Engine описує інтегровані сценарії роботи з системами контролю версій, зокрема Perforce та SVN, а також наводить інші варіанти командної взаємодії [1]. Тому під час аналізу СКВ слід оцінювати не лише функції самого сховища, а й відповідність інструментів конкретному рушію, формату ресурсів і навичкам команди.

Висновки. Специфіка контролю версій у розробці відеоігор визначається поєднанням коду з великими бінарними ресурсами, які часто не підтримують автоматичного злиття. Тому особливого значення набувають збереження історії змін проєкту, узгодження паралельного редагування, вибіркове блокування файлів, зберігання великих об'єктів, інтеграція з рушієм. Git LFS і Perforce реалізують різні механізми роботи з ресурсами, однак аналіз не дає підстав проголошувати одну систему універсально кращою: вибір залежить від типів ресурсів, спільного редагування та організації роботи команди. Визначальними є типи ресурсів, характер спільного редагування та організація роботи команди. Загалом контроль версій у геймдеві доцільно розглядати як частину процесу керування цифровим контентом і координації міждисциплінарної розробки.

Використання інструментів штучного інтелекту. Claude (Anthropic) використано для звірки тверджень із першоджерелами та граматичної перевірки тексту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

[1] Epic Games. (2026). *Source Control*. https://dev.epicgames.com/documentation/unreal-engine/source-control-in-unreal-engine

[2] Epic Games. (2026). *Using Perforce as Source Control.* https://dev.epicgames.com/documentation/unreal-engine/using-perforce-as-source-control-for-unreal-engine

[3] Strange, S. (2025). *Unlocking the research potential of version control logs created by commercial video game development studios* [PhD thesis]. Abertay University. https://rke.abertay.ac.uk/en/studentTheses/unlocking-the-research-potential-of-version-control-logs-created-/

[4] Brian, M.C. (2024). *Git LFS Specification*. GitHub. https://github.com/git-lfs/git-lfs/blob/main/docs/spec.md

[5] Radchenko, M. (2025). *Git LFS File Locking API.* GitHub. https://github.com/git-lfs/git-lfs/blob/main/docs/api/locking.md

Bohutska A.O.

Specific Features of Version Control in Video Game Development

Abstract. *This paper analyses the specific features of version control in video game development, where source code is managed alongside heterogeneous digital assets such as 3D models, textures, audio, animation, and scene files. It discusses versioning large binary objects, limitations of automatic merging, and the role of file locking in team collaboration. Drawing on a doctoral thesis on version-control logs in commercial game development and on technical documentation, the paper examines game-development requirements for change history, synchronization, and collaborative editing.*

Keywords: *version control; video game development; binary assets; collaborative editing; file locking; Git LFS; Perforce.*