Список використаних джерел

1. Деревенко А. М., Ільїна Т. В., Ібрагімова Л. А. Використання цифрових платформ для підвищення якості професійної освіти. *Академічні візії,* 2024. № 31. С. 1–12.

2. Гнатишин М. Аналіз сучасних тенденцій розвитку технологій веб-розробки. *Природничі та гуманітарні науки. актуальні питання :* матеріали І Міжнар. студ. наук.-техн. конф. 2023. С. 132.

3. Зріз ринку цифрової розробки України за 2024 рік. URL: https://it-rating.ua/snapshot-web-development-market-ukraine-2024. (дата звернення:02.11.2025р.).

ОПТИМІЗАЦІЯ ПРОДУКТИВНОСТІ ІГОР, СТВОРЕНИХ З ВИКОРИСТАННЯМ РУШІЯ GODOT ТА C#

Мельник Петро Петрович
здобувач другого рівня вищої освіти, спеціальність Комп'ютерні науки
Тернопільський національний педагогічний університет імені Володимира Гнатюка
melnyk_pp@fizmat.tnpu.edu.ua
Василенко Ярослав Пилипович
викладач кафедри інформатики та методики її навчання
Тернопільський національний педагогічний університет імені Володимира Гнатюка
yava@fizmat.tnpu.edu.ua

У сучасній індустрії відеоігор продуктивність – не просто «приємна властивість», а фундаментальна вимога до успішного проєкту: гравці очікують плавної роботи, швидкого завантаження і відсутності затримок. Особливо це стосується 3D-ігор, мобільних платформ та кросплатформених рішень, де апаратні ресурси можуть бути обмеженими. Враховуючи це, оптимізація ігрового двигуна, рендерингу, скриптової логіки та алгоритмів стає критичною складовою процесу розробки.

Рушій Godot (версії 3.x/4.x) активно розвивається як відкрите рішення для 2D– і 3D-ігор: він пропонує зручну сценно-вузлову архітектуру, підтримку декількох мов програмування (зокрема рідною мовою GDScript та C#) і орієнтований на мультиплатформеність [1].

Однією з сильних сторін Godot є розвинена документація щодо загальних підходів до оптимізації: – наприклад, офіційна стаття «General optimization tips» описує кроки щодо організації проєкту, правильного імпорту ресурсів, налаштування рендерингу, управління світлом і матеріалами [2].

Програмування ігрової логіки на C# у Godot відкриває додаткові можливості: статична типізація, відомі розробникам бібліотеки.NET, кращі засоби під час роботи з більшими проєктами. Зокрема, огляд «Godot C# vs GDScript» звертає увагу на те, що C# «часто перевершує GDScript у виконанні складних обчислень або великих проєктів» – хоча і з певними застереженнями [3].

Отже, тема оптимізації продуктивності при розробці ігор із використанням Godot + C# є надзвичайно актуальною. Вона охоплює кілька вимірів:

– технічний-архітектурний – як правильно структурувати сцени, управляти вузлами (nodes), обирати підходящий рендер-режим;

– алгоритмічний – як мінімізувати витрати на фізику, освітлення, частинки, скасування видимості тощо;

– мовний/скриптовий – як саме мова програмування (C#) і її зв'язок із рушієм впливають на продуктивність, і які практики застосовувати;

– міжплатформений-ресурсний – як оптимізувати проєкти для різних пристроїв, включно з мобільними, або враховувати обмежені апаратні ресурси (наприклад, ARM GPU). Наприклад, блог-публікація «Optimizing 3D scenes in Godot on ARM GPUs» [4] розглядає специфіку мобільної графіки.

У процесі виконання дослідження були використані такі методи: *аналіз і систематизація літературних та електронних джерел* (для вивчення сучасних підходів до оптимізації ігрових рушіїв, особливостей архітектури Godot та принципів роботи мови програмування C#); *порівняльний аналіз* (для зіставлення ефективності різних мов сценаріїв (C# і GDScript) у контексті швидкодії, використання ресурсів і можливостей оптимізації); м*оделювання та експериментальне тестування* (для створення серії тестових ігрових сцен у середовищі Godot та оцінки їхньої продуктивності за різних умов –зміна кількості об'єктів, текстур, джерел світла, фізичних взаємодій тощо); *емпіричний метод вимірювання продуктивності* (із застосуванням вбудованих засобів профілювання Godot (Profiler, Frame Time Graph) та зовнішніх інструментів аналізу (наприклад, Intel GPA, ARM Mobile Studio) для збору статистичних даних про FPS, завантаження процесора та пам'яті); *метод узагальнення та синтезу* (для формулювання практичних рекомендацій щодо підвищення ефективності ігор, розроблених у Godot із використанням C#).

У процесі розробки сучасних відеоігор продуктивність є одним із ключових факторів, що безпосередньо впливає на якість користувацького досвіду. Під продуктивністю розуміють ефективність виконання програмного коду, швидкість відтворення кадрів (FPS), стабільність часу кадру (frame time), а також раціональне використання апаратних ресурсів – центрального процесора (CPU), графічного процесора (GPU) і пам'яті (RAM).

Ігровий рушій Godot Engine є відкритою, багатоплатформною системою для створення 2D– та 3D-ігор. Він використовує сценно-вузлову архітектуру, де кожен об'єкт сцени є вузлом (Node), а вся гра формується як ієрархічна структура цих елементів. Така архітектура забезпечує гнучкість і зручність, але при великій кількості об'єктів може створювати додаткові витрати на обробку сигналів, оновлення вузлів і виклики фізичних процесів.

Godot підтримує кілька мов програмування, серед яких GDScript (власна мова рушія), C#, C++ та VisualScript. Використання C# забезпечує більшу швидкодію при виконанні складних обчислень завдяки компіляції в проміжний байт-код та оптимізаціям CLR (.NET Common Language Runtime). Саме тому C# часто обирають для великих або продуктивно орієнтованих проєктів [3].

Питання оптимізації в Godot детально висвітлені в офіційній документації [2], де підкреслюється, що продуктивність не можна «додати» наприкінці розробки – її потрібно проєктувати з самого початку. Оптимізаційні дії доцільно розглядати у кількох напрямах.

Основне правило – мінімізувати кількість активних вузлів у сцені. Надмірна ієрархічність збільшує кількість оновлень і викликів _process() на кожен кадр. Рекомендується об'єднувати однотипні об'єкти, використовувати Instancing замість

дублювання, а також деактивувати непотрібні вузли під час невидимості (через visible = false або групи «paused»).

Godot використовує різні рендерери (Forward+, Mobile, Compatibility). Для досягнення максимальної продуктивності слід обирати відповідний режим залежно від платформи. У тривимірних сценах продуктивність значною мірою залежить від кількості draw calls, джерел освітлення, складності шейдерів і полігональності моделей.

Методи фрустум-кулінгу (frustum culling) і оклюзійного кулінгу (occlusion culling) дозволяють не рендерити об’єкти, які не видно камері. Використання системи рівнів деталізації (LOD – Level of Detail) зменшує навантаження GPU на великих сценах. Додатково можна оптимізувати матеріали шляхом об’єднання текстур у атласи та зниження їх роздільної здатності без втрати візуальної якості [4].

У C#-скриптах рекомендується мінімізувати створення нових об’єктів під час виконання циклів, щоб уникати частого спрацьовування збирача сміття (Garbage Collector). Доцільно використовувати Object Pooling для повторного використання об’єктів, уникати надмірного використання делегатів і лямбда-функцій у критичних ділянках коду.

Перевагою C# є можливість застосовувати асинхронні методи (async/await) для неблокуючих операцій, наприклад, завантаження ресурсів або мережевих запитів. Це дозволяє скоротити час очікування кадру та підвищити плавність гри.

Godot дозволяє контролювати завантаження ресурсів через механізми streaming (потокового завантаження сцен) і preloading (попереднього завантаження). Для мобільних платформ рекомендується зменшувати обсяг текстур і аудіофайлів, а також уникати зайвих фізичних об’єктів.

На основі проведеного дослідження сформульовано такі рекомендації для підвищення ефективності ігор на Godot із C#:

Використовувати C# для обробки логіки, що вимагає інтенсивних обчислень (штучний інтелект, фізика, генерація карт), залишаючи GDScript для швидких прототипів.

Зменшувати кількість активних вузлів і сигналів у сцені.

Уникати створення об’єктів у циклах – застосовувати пулінг або кешування.

Використовувати профайлер Godot для виявлення «вузьких місць» у рендерингу й логіці.

Оптимізувати ресурси – текстури, звуки, анімації – відповідно до цільової платформи.

У 3D-проєктах застосовувати LOD, baked lighting і фрустум-кулінг.

Регулярно проводити тестування продуктивності під різними налаштуваннями графіки та освітлення.

Отримані результати підтвердили, що вибір мови C# у середовищі Godot може позитивно впливати на загальну продуктивність і стабільність проєкту за умови правильної архітектури сцени та застосування відомих технік оптимізації. Найбільший ефект досягається при оптимізації логіки, рендерингу та управління ресурсами.

Таким чином, оптимізація в Godot є багаторівневим процесом, який поєднує технічні, алгоритмічні та архітектурні рішення. Її ефективність залежить не лише від

використання конкретної мови програмування, а й від грамотного проєктування структури сцени, розподілу навантаження та контролю за споживанням ресурсів.

У ході дослідження були проаналізовані сучасні підходи до підвищення ефективності ігрових проєктів, проведені експерименти з тестовими сценами та сформульовані практичні рекомендації для розробників. Зокрема, визначено ключові чинники продуктивності: кількість активних вузлів у сцені, структура ієрархії, складність рендерингу, обсяг обчислень у скриптах та управління ресурсами; показано переваги використання C# у Godot: реалізації на C# демонструють більш стабільний FPS та зменшене завантаження CPU під час обробки складних логічних та фізичних обчислень, що підтверджено експериментальними вимірюваннями; оцінено ефективність окремих технік оптимізації: застосування LOD, pooling об’єктів, baked lighting і оптимізація ресурсів призводить до відчутного підвищення продуктивності, особливо в 3D-сценах; виявлено компроміс між продуктивністю та зручністю розробки: GDScript забезпечує швидкість прототипування та легшу налагоджуваність, але поступається C# у виконанні обчислювально складних процесів.

Оптимізація продуктивності ігор у Godot є комплексним завданням, що поєднує архітектурні, алгоритмічні та програмні рішення. Використання C# у поєднанні з грамотно спроєктованою сценою, ефективним рендерингом та контролем ресурсів дозволяє створювати стабільні, високопродуктивні ігрові проєкти. Результати дослідження мають практичну цінність для розробників ігор, що прагнуть підвищити продуктивність своїх проєктів на різних платформах.

Список використаних джерел

1. Godot (game engine). URL: https://en.wikipedia.org/wiki/Godot_ %28game_engine %29 (дата звернення: 03.11.2025).

2. Godot Engine 4.5 documentation in English. Performance General optimization tips. URL: https://docs.godotengine.org/en/stable/tutorials/performance/general_optimization.html (дата звернення: 03.11.2025).

3. Kalinda Ch. Godot C# vs Gdscript (How it Works for Developers). 2025. URL: https://ironpdf.com/blog/net-help/godot-csharp-vs-gdscript/ (дата звернення: 03.11.2025).

4. Optimizing 3D scenes in Godot on Arm GPUs. URL: https://developer.arm.com/community/arm-community-blogs/b/mobile-graphics-and-gaming-blog/posts/optimizing-3d-scenes-in-godot-on-arm-gpus (дата звернення: 03.11.2025).

5. Game Optimization Methodology. URL: https://www.intel.com/content/www/us/en/docs/gpa/user-guide/2022-4/game-optimization-methodology.html (дата звернення: 03.11.2025).