Міністерство освіти і науки України
Тернопільський національний педагогічний університет імені Володимира Гнатюка
Фізико-математичний факультет
Кафедра інформатики та методики її навчання

Кваліфікаційна робота

РОЗРОБКА ІГРОВОГО ПРОЄКТУ З ВИКОРИСТАННЯМ РУШІЯ GODOT ТА C#

Спеціальність 122 Комп'ютерні науки
Освітня програма «Комп'ютерні науки»

Здобувача другого (магістерського) рівня вищої освіти
Мельника Петра Петровича
НАУКОВИЙ КЕРІВНИК:
доцент кафедри інформатики та методики її навчання, кандидат біологічних наук
Шмигер Галина Петрівна
РЕЦЕНЗЕНТ:
доцент кафедри комп'ютерних наук Тернопільського національного технічного університету ім. І. Пулюя, кандидат технічних наук
Дмитроца Леся Павлівна

Тернопіль – 2025

АНОТАЦІЯ

Мельник П. П. Розробка ігрового проєкту з використанням рушія Godot та C#. Кваліфікаційна робота на здобуття освітнього ступеня «магістр» зі спеціальності 122 Комп'ютерні науки. ТНПУ ім. В. Гнатюка. Тернопіль, 2025. 63 с.

У кваліфікаційній роботі досліджено методи підвищення продуктивності ігор, створених із використанням рушія Godot та мови програмування C#. Проаналізовано теоретичні основи оптимізації ігрових систем, особливості архітектури Godot і взаємодії рушія з .NET-середовищем. Розглянуто сучасні підходи до оптимізації структури сцени, рендерингу, логіки скриптів, роботи з пам'яттю та ресурсами.

У рамках експериментальної частини розроблено серію тестових сцен, у яких порівняно продуктивність реалізацій на GDScript і C#. Вимірювання проводилися за допомогою профайлера Godot, Intel GPA та інструментів аналізу графічної підсистеми. Результати показали, що використання C# забезпечує підвищену ефективність обчислень і стабільність FPS у ресурсомістких ігрових проєктах. На основі отриманих даних сформульовано практичні рекомендації щодо оптимізації логіки, рендерингу та управління ресурсами в Godot.

Ключові слова: Godot, C#, оптимізація та продуктивність ігор, рендеринг, профілювання, GDScript, ігровий рушій, .NET.

ABSTRACT

Melnyk, P. P. Development of a game project using the Godot engine and C#. Qualification work for obtaining a Master's degree in Computer Science, speciality 122. Ternopil Volodymyr Hnatyuk National Pedagogical University. Ternopil, 2025. 63 p.

This thesis explores methods for improving the performance of games created using the Godot engine and the C# programming language. It analyses the theoretical foundations of game system optimisation, the features of Godot's architecture, and the engine's interaction with the .NET environment. It considers modern approaches to optimising scene structure, rendering, script logic, memory and resource management.

As part of the experimental part, a series of test scenes were developed to compare the performance of implementations in GDScript and C#. Measurements were taken using the Godot profiler, Intel GPA, and graphics subsystem analysis tools. The results showed that using C# provides increased computational efficiency and FPS stability in resource-intensive game projects. Based on the data obtained, practical recommendations for optimising logic, rendering, and resource management in Godot were formulated.

Keywords: Godot, C#, game optimisation and performance, rendering, profiling, GDScript, game engine, .NET.

ЗМІСТ

ВСТУП

Актуальність теми дослідження. Поява нових технічних рішень в індустрії відеоігор перетворює її на одну з сфер цифрової економіки. Глобальні аналітичні звіти демонструють стабільне зростання ринку, а кількість користувачів цифрових ігор налічує понад два мільярди осіб. Відеоігри набули статусу не лише одного з провідних сегментів сфери розваг, а й важливого технологічного, освітнього та соціального інструмента. Цифрові ігрові середовища інтенсивне поширення у системи віртуальної та доповненої реальності, професійні тренажери, медичні симулятори, системи корпоративного навчання й освітні платформи, що засвідчує значний потенціал їхнього міждисциплінарного застосування.

Технологічний прогрес, удосконалення графічних рушіїв, впровадження штучного інтелекту, адаптивних систем логіки та високопродуктивних мов програмування сприяли зміні підходів до розробки ігрового програмного забезпечення. Цифровий процес створення ігор ґрунтується на комплексному використанні інтегрованих середовищ розробки, систем 3D-моделювання, анімаційних технологій, інструментів оптимізації продуктивності та бібліотек для підтримки складної ігрової логіки. У таких умовах розробникам необхідно володіти актуальними інструментами проєктування ігрових середовищ, що забезпечують високий рівень продуктивності та гнучкості.

Godot Engine вирізняється серед сучасних технологічних платформ завдяки відкритій архітектурі, підтримці модульності, кросплатформенності, інтеграції алгоритмів оптимізації та можливості використання мови C#. Поєднання цих характеристик робить рушій придатним як для навчальних і дослідницьких завдань, так і для професійної розробки інтерактивних ігрових систем.

Дослідження теми розробки ігрових проєктів із застосуванням рушіїв нового покоління, зокрема Godot Engine, представлено у працях окремих науковців і практиків, серед яких С. Альхімович, М. Ангел, С. Андреєв, Р. Базюк, М. Бойченко, Д. Вербовецький, С. Вітвицька, Л. Віната, О. Воєвода, Д. Гольфельд, О. Долгополов, І. Доброскок, М. Жук, М. Плєхавська-Вуйцик, М. Ранавера, І. Сергородцев, Д. Сергеєв, Д. Стрига, Д. Цикал, С. Ярвенпа. Актуальність даної теми визначається потребою в дослідженні практичних і технічних аспектів створення ігрового проєкту з використанням доступних інструментів та рушія Godot, а також необхідністю аналізу можливостей оптимізації графічного рендерингу, моделювання поведінкових систем і організації внутрішньої логіки гри, що і зумовлює вибір теми дослідження «**Розробка ігрового проєкту з використанням рушія Godot та C#».**

Мета дослідження: обґрунтувати технологічні підходи до оптимізації графічної, логічної та структурної складових гри, розробити ігровий проєкт на основі Godot Engine із використанням мови C#.

Для досягнення поставленої мети сформульовано такі **завдання**:

1. Визначити роль мови програмування C# у контексті розробки ігор.

2. Дослідити функціональні можливості Godot Engine для створення інтерактивного тривимірного середовища.

3. Створити ігровий проєкт із застосуванням мови програмування C#.

4. Розробити базові ігрові механіки відповідно до концепції гри.

Об'єкт дослідження – ігровий програмний продукт, розроблений із використанням Godot Engine та C#.

Предмет дослідження – технології, архітектурні підходи, інструменти оптимізації та програмування, що застосовуються для створення ігрових систем у середовищі Godot.

Для досягнення мети та реалізації поставлених завдань було використано комплекс взаємопов'язаних методів дослідження: теоретичних методів (аналіз, порівняння, узагальнення), логіко-методологічних методів,

практично-емпіричних методів (проєктування, моделювання, експериментальна перевірка рішень).

Наукова новизна полягає у практичному обґрунтуванні можливостей інтеграції C# із системами оптимізації Godot Engine для розробки масштабованого ігрового середовища.

Практичне значення: визначається тим, що матеріали кваліфікаційної роботи можуть бути використані в галузі Комп’ютерні науки здобувачами закладів освіти, початківцями та фахівцями сфери ІТ. Результати розробницької діяльності, а саме ігрового проєкту можуть бути використані як засіб для розваги, зняття стресу та емоційного розвантаження.

РОЗДІЛ 1
СУЧАСНІ ІНСТРУМЕНТИ СТВОРЕННЯ ІГРОВОГО КОНТЕНТУ

1.1 Godot Engine як рушій для розробки сучасних ігрових проєктів

З розвитком технологій, зміною потреб гравців та зростанням конкурентного тиску, розробники змушені використовувати передові інструменти, які не лише оптимізують процес розробки, а й створюють більш інтерактивний, захопливий ігровий досвід. Сьогодні розробка ігор набула комплексного міждисциплінарного характеру, що об’єднує різноманітні дисципліни, від програмування до дизайну та звукового оформлення.

Однією з ключових тенденцій є використання потужних ігрових рушій, таких як Unreal Engine, Unity та інші, що забезпечують багатий набір інструментів для створення високоякісної графіки, реалістичної фізики та складних анімацій. Завдяки такій платформі навіть незалежні розробники отримали доступ до технологій, які раніше були доступні лише великим студіям з багатомільйонними бюджетами.

Крім того, хмарні послуги і засоби колаборативної роботи відкрили нові горизонти в процесі організації розробки. Інструменти для роботи з версіями, як-от Git, системи управління завданнями та сервісами на основі хмарних обчислень, дозволяють командам з різних куточків світу одночасно працювати над одним проєктом без затримок та проблем із синхронізацією.

Не менш важливе значення приділено технологіям штучного інтелекту та машинного навчання, які впроваджуються для створення більш адаптивних та інтелектуальних ігрових персонажів, прогнозування поведінки гравців та персоналізації ігрового процесу, що дозволяє створювати динамічні світи, які реагують на дії користувача в реальному часі, розширивши спектр творчих та інженерних підходів.

Галузь комп’ютерних ігор відзначає безпрецедентне зростання та еволюцію, що створює нові проблеми для розробників у всьому світі.

Основною проблемою, яка стоїть перед сучасними розробниками, є необхідність створення високоякісного ігрового контенту в умовах жорсткої конкуренції, стрімких змін у технологіях та зростання очікувань користувачів. З одного боку, користувачі прагнуть бачити ігри з реалістичною графікою, захопленим сюжетом і високою інтерактивністю, з іншого боку, самі розробники стикаються з обмеженнями в часі, ресурсах і знаннях, більшою мірою для створення такого контенту.

Складність сучасних ігор, що включають у себе роботу з величезними обсягами даних, інтеграцію різноманітних технологій та підтримку багатьох функцій, вимагає використання спеціалізованих інструментів, які можуть оптимізувати процес розробки та мінімізувати витрати. Проте, не всі інструменти однаково ефективні для різних типів проєктів, і вибір оптимального набору технологій часто стає значною проблемою для розробників, особливо для менших команд і незалежних студій.

Актуальність даної проблеми полягає у тому, що стрімке зростання ігрової індустрії перетворює її на одну з провідних галузей цифрової економіки. За даними останніх досліджень, ринок відеоігор генерує мільярд доларів доходу щороку, а кількість гравців у всьому світі перевищує кілька мільярдів осіб. У таких умовах здатність розробників адаптуватися до нових вимог і ефективно використовувати сучасні інструменти для створення контенту стає вирішальним фактором успіху.

Технологічний прогрес постійно змінює галузевий ландшафт розробки ігор, пропонуючи нові можливості для оптимізації процесів та підвищення продуктивності. Тому дослідження та впровадження нових інструментів є критичним місцем для підтримки конкурентоспроможності розробників.

Крім того, сучасні гравці очікують високої якості продукту, який вимагає від розробників швидкої адаптації до нових тенденцій і технологій. Враховуючи ці аспекти, дослідження сучасних інструментів для створення ігрового контенту є не лише актуальним, а й необхідним для підтримки інноваційного розвитку промисловості.

Виокремимо, що ігровий рушій це – програмне середовище (платформа), призначене для створення комп'ютерних ігор та інтерактивних застосунків, яке надає готові інструменти для роботи з графікою, фізикою, звуком, логікою гри та керуванням ресурсами. Ключовим рушієм розробки сучасних ігрових проєктів є – Godot Engine.

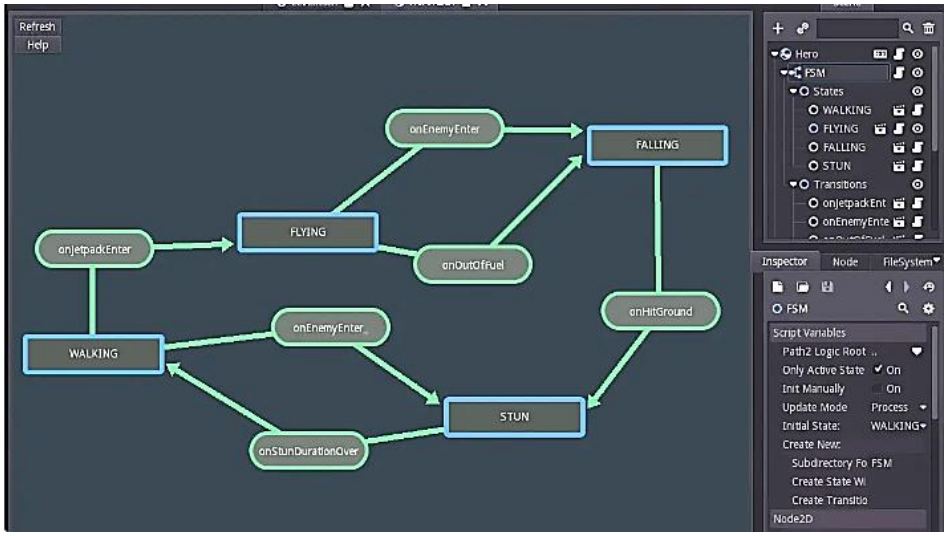

Рис. 1.1. Філософія дизайну Godot (джерело: авторська розробка)

Godot Engine – це відкритий ігровий рушій, який набирає популярність завдяки своїй гнучкості та простоті у використанні. Він підтримує як 2D, так і 3D розробку ігор, що робить його універсальним вибором для розробників різного рівня. Godot пропонує власну мову програмування GDScript , яка нагадує Python і дозволяє швидко освоювати основи розробки.

Д. Джуга та С. Мартинюк визначили структуру роботи в Godot Engine, що містить такі основні складові:

- «сцена – це один або певна кількість (структура) вузлів, які утворюють дерево і зберігаються окремим файлом;
- скрипт – код, написаний на доступних для використання мовах рушія, під'єднаний до вузла сцени, інтегрований у файл сцени або збережений окремо;
- вузол – це об'єкт, наділений власними властивостями й унікальним функціоналом. Їх можна поділити на такі класи:

– «Node2D» – вузли для створення 2D ігрових сцен;

– «Node3D» – вузли для створення 3D ігрових сцен;

– «Control» – вузли для створення користувацького інтерфейсу;

– інші вузли загального використання [9, с. 156].

Одним із головних переваг Godot є його легкість і невеликі системні вимоги, що робить його ідеальним для інді-розробників та невеликих студій. Крім того, Godot має інтегрований редактор сцени, підтримує роботу з широким спектром платформ та надає набір інструментів для розробки інтерфейсу користувача, анімації та шейдерів. Завдяки своїй відкритості Godot дозволяє розробникам змінювати й адаптувати рушій під власні потреби.

«Спочатку Godot був розроблений власними силами аргентинської ігрової студії. Його розробка розпочалася у 2001 році, а у 2014 році» [2], його випустили з відкритим вихідним кодом під ліцензією MIT (Массачусетського технологічного інституту). Ігровий рушій Godot створили два розробники з Аргентини – Д. Лінітський та А. Манзур. «Понад десять років Годо залишався суворо охоронюваною таємницею, використовувався виключно для внутрішніх проєктів у компанії Okam Studio» [50].

Розробники рушія, обрали назву Godot як посилання на п’єсу С. Бекета «У очікуванні Ґодо». У цій п’єсі персонажі чекають на таємничого Ґодо, який так і не з’являється (рис. 1.2).

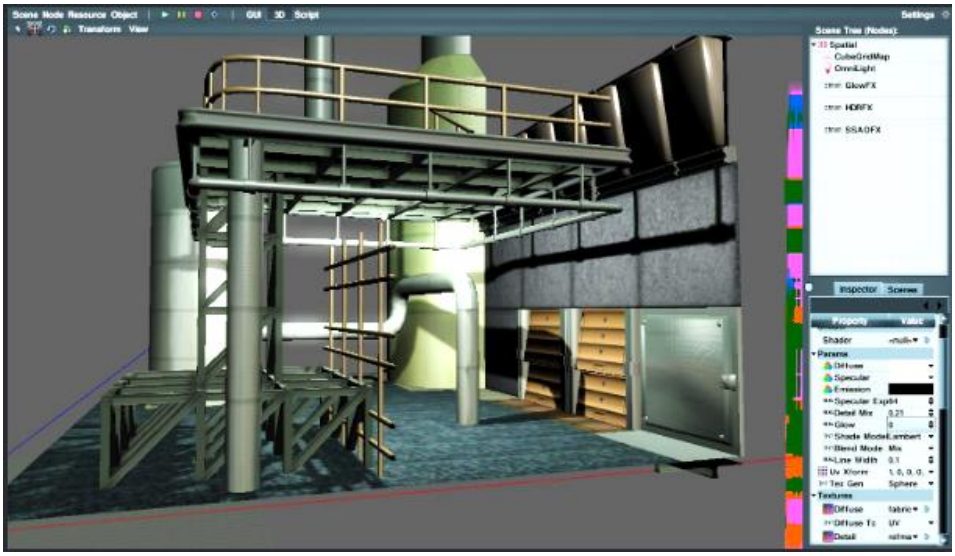

Рис. 1.2. Скріншот Godot початку 2010-х років до відкриття вихідного коду

Рис. 1.3. Ігри, створені з Godot до відкритого коду

«Однак зручність використання залишалася основною метою. Користувачам досі потрібно було пройти багато кроків, щоб зробити те, що було простіше в інших двигунах» [41] – зазначає Д. Лініцький. Тому розробка тривала допоки не вийшов Godot 2.0, а у 2016 – версія 2.1. Надалі рушій стрімко розвивася, і уже в 2018 році випущений модернізований Godot 3.0. Уже у 2023 вийшла новіша версія Godot 4.0. На сьогоднішній день найновіша версія випуску – це Godot 4.4.1. випущений у 2025 році (рис. 1.4).

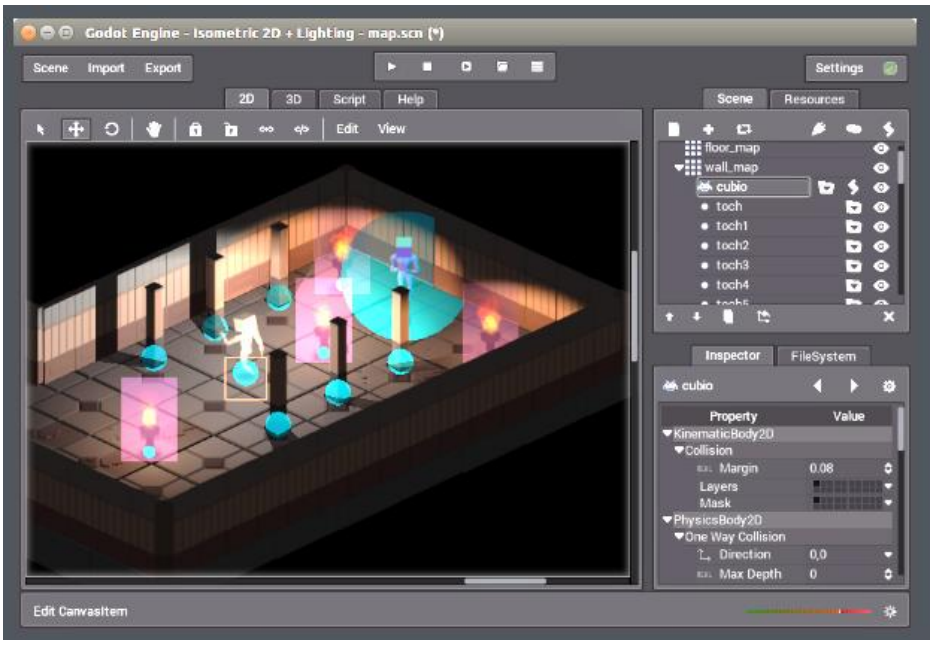

Рис. 1.4. Godot 1.0, випущений у грудні 2014 року

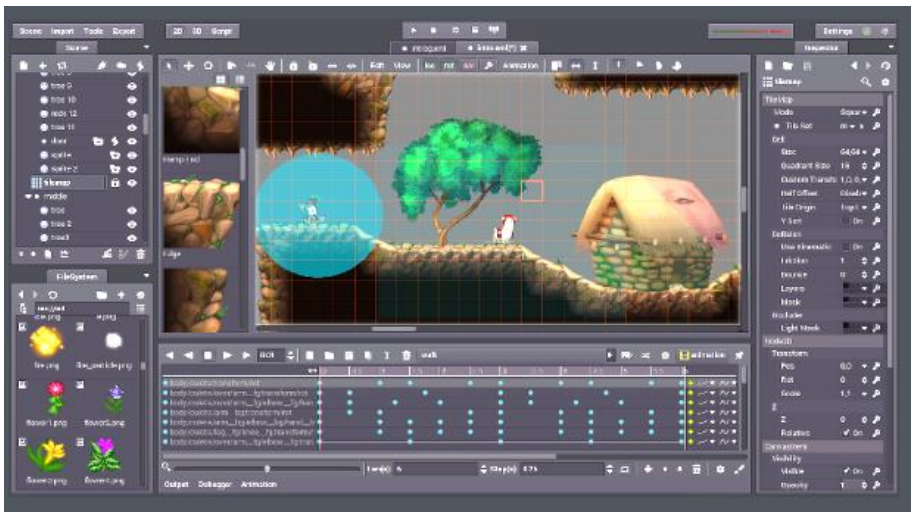

Рис. 1.5. Godot 2.0, випущений у лютому 2016 року

Нова архітектура візуалізації також дозволить компаніям, які працюють над консольними портами, ефективніше переносити движок і зможе запропонувати користувачам можливість запускати ігри на найпопулярніших ігрових консолях (рис. 1.6).

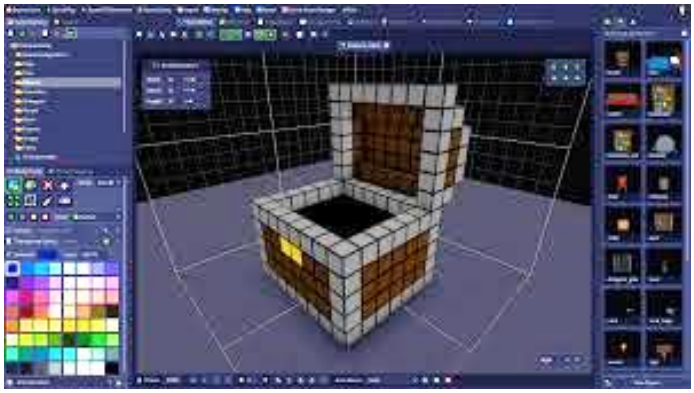

Рис. 1.6. Godot Game Engine

Згідно з результатами Google trends [37] за п’ять років (2020-2025) популярність рушія зростає (рис. 1.7).

Рис. 1.7. Інтерес користувачів у світі (2 квітня 2020 – 2 лютого 2025)

Також представлені країни з найбільшою кількістю пошукових запитів по Godot, лідерами яких є Китай, Швеція, Фінляндія, Естонія та Франція (рис. 1.8). Визначені результати показують, що рушій Godot Engine користується попитом у всьому світі.

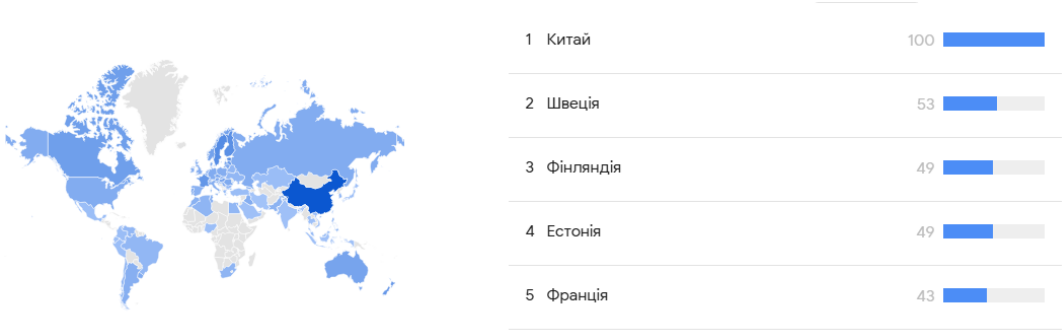

Рис. 1.8. Популярність за регіонами

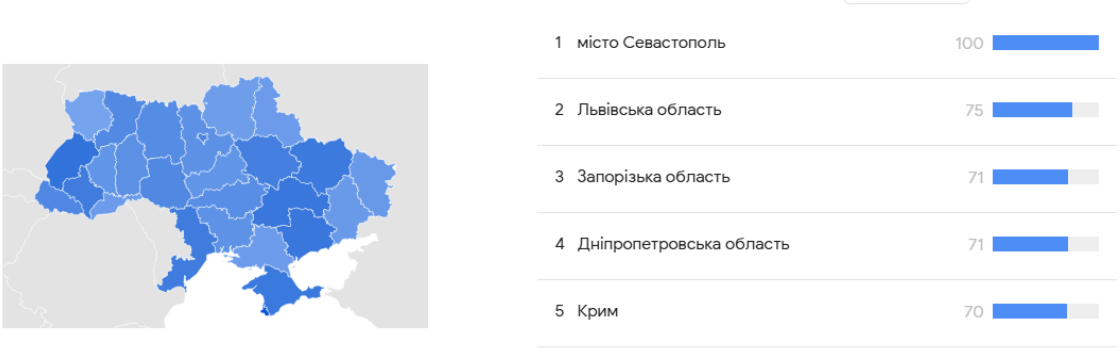

Рис. 1.9. Популярні запити за територіальними одиницями

Технічно розробка ігор безупинно розвивається, що зумовлює потребу у використанні спеціалізованих технологічних засобів для оптимізації процесу створення якісного ігрового продукту. Ефективна організація розробки вимагає комплексного підходу, де кожен етап – від проєктування ігрової концепції до реалізації і тестування фінальної версії – забезпечується відповідними інструментами.

Сучасні інструменти для створення ігрового контенту можна розділити на кілька ключових категорій, які охоплюють усі етапи розробки ігор, тому категорії включають ігрові рушії, інструменти для роботи з графікою, звуком, анімацією, штучним інтелектом, а також платформи для тестування та дебагу. Кожна з цих категорій відіграє важливу роль у створенні інтерактивного ігрового досвіду.

Ігрові рушії. Ігрові рушії є основою будь-якої гри, забезпечуючи базову інфраструктуру для її функціонування. Найпопулярнішими серед сучасних рушіїв є Unreal Engine, Godot Engine та Unity, які пропонують широкий спектр для створення як 2D, так і 3D ігор.

Unreal Engine вирізняє всі можливості для високоякісної графіки та реалістичної фізики, що робить його популярним вибором для розробки AAA-проєктів. З іншого боку, Unity пропонує більшу гнучкість і простоту використання, що дозволяє створювати як невеликі інді-проєкти, так і складні багатоплатформенні ігри.

Рушії підтримують додатків та плагіни, що значно розширюють їх функціональність, що дозволяють адаптуватися до різноманітних потреб розробників (рис. 1.10).

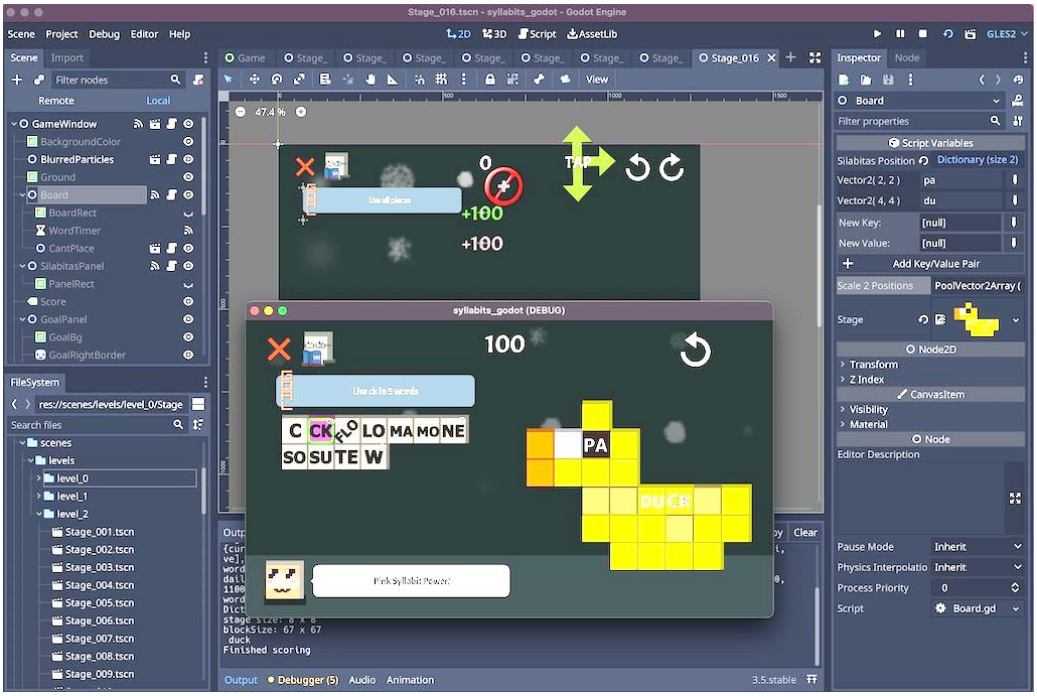

Рис. 1.10. Reasons for a solo dev to love Godot Engine

Інструменти для роботи з графікою. Графічний контент є ключовим елементом видимої привабливості гри. Для створення 2D графіки розробники використовують такі програми, як: Adobe Photoshop, Aseprite, Krita, Affinity Photo, Pyxel Edit.. Для 3D моделювання та скульптування: Blender (безкоштовний та з відкритим кодом), Autodesk Maya, Autodesk 3ds Max,

ZBrush, Substance Painter (текстурування), Substance Designer (створення матеріалів). Інтеграція цих інструментів з ігровими рушіями дозволяє ефективно імпортувати та досягати графічні елементи в ігровому середовищі.

Анімація і кінематика. Інструменти для анімації, такі як Autodesk MotionBuilder та Mixamo, надають розробникам можливість створювати складні рухи персонажів та об’єктів. Надають можливість оживити персонажів, об’єкти та елементи інтерфейсу через створення послідовності зображень або рухів 3D-моделей. Щодо 2D анімації використовуються: Spine, DragonBones, Adobe Animate. Інструмент MotionBuilder пропонує широкий спектр функцій для захоплення руху та роботи з анімаціями, тоді як Mixamo автоматизує процес ригінгу та анімації 3D-моделей, значно скорочуючи час, необхідний для створення реалістичних рухів.

Звукові інструменти. Звук є невід’ємною частиною ігрового досвіду, а інструменти для роботи зі звуком, що дозволяють розробникам інтегрувати комплексні звукові ефекти та музичні доріжки. Наприклад, цифрові аудіо робочі станції (DAWs) такі як: Ableton Live, Logic Pro X, FL Studio, Reaper (доступний), Audacity (безкоштовний). Використовуються програми для створення звукових ефектів: Bfxr (безкоштовний), Serum, Massive, та інструменти для інтеграції звуку в гру: Wwise, FMOD.

Платформи забезпечують можливості для динамічного звуку, який змінюється залежно від дій гравця, що забезпечує глибину ігрового процесу. Забезпечують створення, редагування, інтеграцію звукових ефектів, музики та озвучення в гру, що значно збагачує ігровий досвід.

Штучний інтелект та машинне навчання. Сучасні ігри де частіше виконують штучний інтелект для створення реалістичної поведінки далеких персонажів та адаптації гри до дій гравця. Допомагають створювати розумних противників, неігрових персонажів (NPC) зі складною поведінкою, а також реалізовувати системи прийняття рішень та навігації в ігровому світі. Інструменти для роботи зі штучним інтелектом, такі як Behavior Trees в Unreal Engine або машинне навчання в Unity ML-Agents, дозволяють

створити складні алгоритми, які можна навчатися та адаптувати до різних сценаріїв в ігровому світі, що забезпечує більш глибокий і реалістичний ігровий досвід.

Використовуються зовнішні бібліотеки та фреймворки такі як TensorFlow, PyTorch – можуть використовуватися для навчання в іграх, хоча це поки не є мейнстримом.

Інструменти для тестування та дебагу. Тестування є наступним етапом у процесі розробки ігор, і інструменти для автоматизованого тестування та дебагу, такі як Visual Studio та JetBrains Rider, допомагають розробникам виявляти та виправляти помилки на ранніх стадіях розробки, що підвищує загальну стабільність і якість кінцевого продукту.

Godot має широкий спектр вбудованих інструментів і «головними компонентами, які необхідні для роботи є графіка та сам код гри, який здійснює рух супротивників по карті. Графіка є індивідуальним компонентом для будь-якого ігрового рушія, оскільки потреби розробників відмінні, існують потреби реалістичного зображення, цим обумовлено використання всіх можливостей цільової платформи, разом з тим іншим розробникам достатнє використання простих двовимірних зображень на однокольоровому фоні» [29; с. 373–374].

Впровадження сучасних інструментів значно змінило підхід до створення ігор. Завдяки їм, розробники мають змогу створювати складніші та більш інтерактивні проєкти, скорочуючи час і витрати на розробку. Інструменти надають змогу меншим командам конкурувати з великими студіями, створюючи якісні продукти, які знаходять споживача свого у всьому світі.

Godot Engine зарекомендував себе як потужний, надійний і доступний інструмент для розробки ігор, що підходить як для новачків, так і для досвідчених розробників. Відкритий код рушія і підтримка різних платформ використовуються для створення широкого спектру ігрових проєктів, включаючи 2D, 3D і проєкти навіть з використанням штучного інтелекту та

машинного навчання. Завдяки легкості освоєння та високій модульності Godot стає особливо популярним серед інді-розробників та освітніх закладів, які хочуть навчатися в основі ігрової розробки. Виділимо основні переваги Godot, що включають:

- безкоштовний та з відкритим кодом, відсутність ліцензійних платежів та повна свобода модифікації рушія під власні потреби;
- інтуїтивно зрозумілий інтерфейс, тому що Godot має якісно організований та орієнтований на користувача інтерфейс, що полегшує навчання та розробку;
- підтримка різних мов програмування: Godot підтримує власну скриптову мову GDScript (схожий на Python), C# та C++. Завдяки GDNative можна інтегрувати й інші мови;

«GDScript – основна сценарна мова Godot, розроблена спеціально для розробки ігор. Вивчіть синтаксис, функції та найкращі практики GDScript, включаючи змінні, функції, керуючі структури, класи та успадкування» [20].

- кросплатформність рушія Godot дозволяє експортувати проєкти на різні платформи, включаючи Windows, macOS, Linux, Android, iOS, HTML5 та консолі (через сторонніх видавців);
- навколо Godot сформувалася спільнота розробників, які діляться знаннями, створюють доповнення та допомагають у вирішенні проблем;
- підтримка сучасних технологій рендерингу, такі як нормальне відображення, спекулярність, динамічні тіні, глобальне освітлення (як запечене, так і динамічне), а також пост-ефекти (bloom, глибина різкості, HDR тощо);
- потужний набір вбудованих інструментів, що включає редактори для 2D та 3D графіки, анімації, тайлових карт, рівнів, шейдерів та багато іншого, що дозволяє розробляти гру повністю в одному середовищі;
- рушій розвивається, систематичне оновлення до нових версій з покращеннями та новими функціями. Остання версія, Godot 4, отримала значні оновлення в системі рендерингу, фізики та інших областях;

- унікальна система вузлів та сцен, архітектура, заснована на вузлах, забезпечує гнучкість та модульність при створенні ігрової логіки та рівнів. Кожен елемент гри є вузлом, який можна комбінувати в складніші сцени.

«Рушій Godot Engine для розробки мережевої частини гри має невелику кількість варіантів. Наразі існують два доступні варіанти: використання вбудованого функціоналу – Godot Engine має вбудоване API для роботи з мережевими функціями через HTTP, UDP, TCP та SSL протоколи; створення власної мережевої системи на мові C#. Недоліком використання Godot Engine при розробці саме мережевих рішень є відсутність готових рішень від сторонніх розробників. Оскільки рушій є відносно новим і немає уваги зі сторони великих компаній, таких як Photon, DarkLib або PlayFab, які декілька років постачають стабільні рішення для інших ігрових рушіїв» [29; с. 373]. Тому виникають обмеження, які варто враховувати:

- документація може бути менш вичерпною в деяких аспектах, хоча документація Godot постійно покращується, в деяких нішевих питаннях вона може бути менш детальною порівняно з більш зрілими рушіями;
- менша кількість вакансій для розробників Godot;
- менша кількість готових асетів та плагінів у порівнянні з Unity та Unreal Engine, екосистема асетів та плагінів для Godot є меншою, хоча вона постійно зростає;
- можливі складнощі з оптимізацією для дуже вимогливих 3D-проєктів, хоча Godot 4 значно покращив рендеринг, для створення фотореалістичних ігор з відкритим світом може знадобитися більше зусиль з оптимізації порівняно з Unreal Engine.

Хоча Godot все ще вважається відносно новим рушієм у порівнянні з гігантами, адже вже є приклади успішних сучасних ігор, створених на ньому, що демонструють його можливості. Приклади сучасних ігор, розроблених на Godot Engine:

- Bounty of One (швидкий арена-шутер з елементами roguelike);
- Brotato (гра в жанрі roguelite arena shooter);

- Cassette Beasts (рольова гра з покроковими боями, де користувач збирає та трансформується в фантастичних істот, записаних на касети);
- Cruelty Squad (кіберпанковий шутер від першої особи з хардкорним геймплеєм та унікальним візуальним стилем);
- Dome Keeper (атмосферна гра про оборону купола від хвиль монстрів);
- Godot Engine Demos and Templates (створює різноманітні демонстраційні проєкти та шаблони, які показують можливості рушія в 3D (наприклад, демонстрації освітлення, фізики, процедурної генерації);
- High Tail Hall (пригодницька гра в жанрі «point-and-click»);
- Sokobos (мінімалістична головоломка, заснована на механіці Sokoban);
- Sonic Colors: Ultimate (ремастер відомої гри від SEGA);
- The Case of the Golden Idol (детективна гра, де користувач розслідує загадкові вбивства, аналізуючи сцени злочинів та розкриваючи зв’язки між персонажами).

Також багато інших інді-ігор різних жанрів, доступних на платформах itch.io та Steam. У рушії Godot Engine, використовуються сценарії (Scripts) – це файли, які містять код, написаний однією з підтримуваних мов програмування (найчастіше GDScript), і прикріплюються до вузлів (Nodes) в ігровій сцені. Їх основна мета – визначати поведінку, логіку та функціональність цих вузлів. «Сценарії – це фундаментальний аспект розробки ігор у Godot Engine, що дозволяє визначати поведінку, логіку та взаємодію ігрових об’єктів за допомогою коду. За допомогою мов сценаріїв Godot, таких як GDScript і візуальних сценаріїв, ви можете створювати складні та чутливі ігрові механізми» [20].

Цифрові ігри характеризується розширення прикладного застосування комплексних рушіїв розробки, які інтегрують засоби графічного моделювання, програмування, анімації та оптимізації продуктивності в єдиному технологічному середовищі.

Особливе місце серед доступних платформ посідає Godot Engine, що є потужним, гнучким та безкоштовним інструментом, який цілком придатний для розробки сучасних ігрових проєктів, його унікальна архітектура, підтримка різних мов програмування та постійний розвиток роблять його привабливим вибором як для початківців, так і для досвідчених розробників. Хоча він може мати певні обмеження порівняно з більш зрілими рушіями, його переваги, особливо в плані доступності та свободи, роблять його перспективним варіантом для створення різноманітних ігор. Загалом, Godot Engine є чудовим вибором для сучасних ігрових проєктів, особливо для тих, хто шукає гнучкість і доступність у розробці ігор.

Поєднання означених можливостей забезпечує ефективну реалізацію інтерактивних ігрових систем різного рівня складності та створює передумови для використання рушія як у професійній розробці, так і в навчально-дослідницькій діяльності. Проведений огляд підтверджує доцільність обраних технологічних рішень і формує теоретичну основу для подальшої практичної реалізації ігрового проєкту.

Насамкінець, сучасні інструменти для створення ігрового контенту дають змогу значно скоротити час на розробку, підвищити якість ігрових продуктів і, головне, реалізувати найсміливіші ідеї розробників. У результаті технологічні інновації продовжують змінювати індустрію комп’ютерних ігор, формуючи нові стандарти якості та можливостей ландшафту.

1.2 Переваги використання мови програмування C# у створенні ігрових механік

В розробці відеоігор, мови програмування є ключовими, визначаючи як технічні можливості продукту, так і швидкість створення контенту. Особливої популярності набуло використання мови програмування C# (C-Sharp) у створенні ігрових механік, що пояснюється її універсальністю,

високим рівнем абстракції, об’єктно-орієнтованим підходом і інтеграцією з популярними ігровими рушіями.

Мова програмування C# давно зарекомендувала себе як один із найпотрібніших інструментів у сфері розробки програмного забезпечення, а в контексті ігрової сфери де її значення лише зростає. Використання C# у створенні ігрових механізмів стає дедалі популярним завдяки його високій продуктивності, гнучкості та підтримці, що є ефективним для складних ігрових систем. Мова дозволяє розробникам ефективно створювати ігри різних жанрів і рівнів складності, інтегруючи сучасні технології та алгоритми, які роблять ігровий досвід більш захоплюючим і реалістичним.

Крім того, C# забезпечує високий рівень зручності завдяки своїй синтаксичній простоті та багатству інструментів для накладання та оптимізації коду, що особливо важливо у сфері ігрової інженерії, де продуктивність коду може впливати на ігровий процес і загальне задоволення користувача. Мова підтримує сучасні парадигми програмування, такі як асинхронність і багатопоточність, що дозволяє ефективно використовувати ресурси пристроїв і забезпечити використання плавного ігрового досвіду навіть на менш потужному обладнанні.

«Багатопоточність – це техніка в програмуванні, яка дозволяє одному процесу виконувати кілька завдань одночасно, підвищуючи тим самим продуктивність та чутливість» [2].

Співробітник Bell Labs Д. Рітчі створив мову C у 1972 році під час співпраці з К. Томпсоном над операційною системою UNIX. Мова була розроблена як інструмент для програмістів-практиків [17]. Мова C швидко стала популярною і однією з найважливіших мов програмування. Відповідно поділяються на рівень складності (табл. 1.1).

Таким чином, модернізована версія C# стає інструментом для фахівців у сфері комп’ютерної ігрової інженерії, надаючи їм можливість розробляти складні, продуктивні та багато функціональні ігри. Завдяки своїм численним перевагам ця мова продовжує формувати майбутні ігрові розробки,

дозволяючи створювати інноваційні проєкти, які високо очікуються сучасними гравцями. «C# – об’єктно-орієнтована мова програмування. Назва читається як «сі шарп». Шарп – нота до-дієз у музиці (дієз = підвищення на пів тон). Тобто C# – це оновлення, покращення мови C» (табл. 1.1) [26].

Таблиця 1.1

Рівні складності мов програмування

Рівень	Назви мови програмування
Високий	Ada Modula-2 Pascal Cobol Fortran Basic
Середній	Java C# C++ C
Низький	Marco-assembler Assembler

Наприкінці 1990-х років, коли Microsoft розпочала розробку нової мови програмування, основною метою було створення сучасної та потужної мови, здатної спростити розробку програм для Windows і .NET Framework. C# було презентовано світові у 2000-х рр. разом із випуском першої версії .NET Framework. Мову розробила команда інженерів під керівництвом А. Гейлсберга, який уже мав досвід роботи над відомими проєктами Delphi і Turbo Pascal [23].

Уже тоді мова програмування налічувала класи, інтерфейси, наслідування, винятки (Exceptions). З моменту випуску і до сьогодні мова програмування стрімко розвивалася. Ми визначили ключові зміни, а саме у:

- 2005 р.: додали Generics (узагальнене програмування, як шаблони в C++); З’явилися анонімні методи та ітератори (yield return); введено nullable types (для роботи з null).

- 2007 р.: з’явився LINQ (Language Integrated Query) – мегазручні запити до колекцій; анонімні типи, лямбда-вирази, автоматичні властивості (auto-properties).
- 2010 р.: додано dynamic тип для роботи з нестрого типізованими об’єктами; optional parameters і named arguments; полегшено інтеграцію з COM, офісними застосунками (Word, Excel).
- 2012 р.: async/await для асинхронного програмування; код став значно простішим при роботі з потоками та операціями вводу/виводу.
- 2015 р.: string interpolation ($"Hello {name}"); expression-bodied members (короткий запис методів).
- 2017 р.: tuples та deconstruction (var (x, y) = GetPoint();); pattern matching (перевірка типів та значень за шаблоном); local functions – функції всередині методів.
- 2019 р.: nullable reference types (сильніше управління null); async streams (await foreach); default interface methods.
- 2020 р.: records (immutable типи для даних, автоматично створюються Equals, GetHashCode, ToString); init-only properties; top-level programs (програми без Main-методу явно в коді) (рис. 1.11).

```
public record Person(string Name, int Age);

var person = new Person("Anna", 25);
Console.WriteLine(person); // Виведе: Person { Name = Anna, Age = 25 }
```

Рис. 1.11. Records (immutable типи даних)

- 2021 р.: global using directives; file-scoped namespaces (менше коду для оголошення простору імен).
- 2022 р.: raw string literals (рядки без потреби екранізації); list patterns, required properties (рис. 1.12).

```
object obj = 42;

if (obj is int number && number > 0)
{
    Console.WriteLine($"Позитивне число: {number}");
}
```

Рис. 1.12. Pattern Matching (джерело: авторська розробка)

```
string GetDayType(string day) => day switch
{
    "Saturday" or "Sunday" => "Weekend",
    "Monday" => "Worst Day",
    _ => "Workday"
};
```

Рис. 1.13. Pattern Matching (switch) (джерело: авторська розробка)

•2023: primary constructors для класів і структур; collection expressions ([1; 2; 3]) для списків без ініціалізації через new List<>; більше зручності для шаблонного (template) програмування (рис. 1.14).

```
public class Circle(double radius)
{
    public double Radius { get; set; } = radius;

    public double Area() => Math.PI * Radius * Radius;
}

var circle = new Circle(5);
Console.WriteLine(circle.Area()); // Виведе площу круга
```

Рис. 1.14. Primary Constructors (C# 12) (джерело: авторська розробка)

```
var numbers = [1, 2, 3, 4, 5];
```

Рис. 1.15. Collection Expressions (outdated version) (джерело: авторська розробка)

```
var numbers = new List<int> { 1, 2, 3, 4, 5 };
```

Рис. 1.16. Collection Expressions (new version) (джерело: авторська розробка)

Згідно з даними Statista [44] за 2024 рік (рис. 1.17) було висвітлено результати найбільш використовувані мови програмування серед розробників у всьому світі:

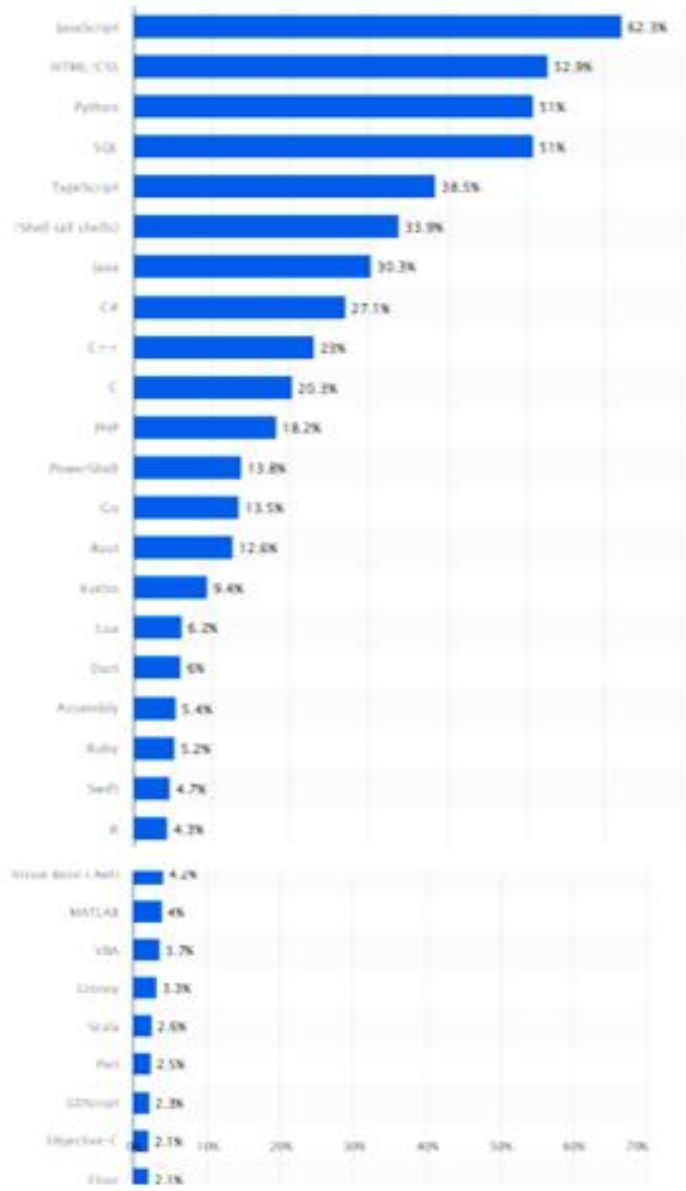

Рис. 1.17. Рейтинг мов програмування з даними Statista [41]

Дані свідчать про те, що JavaScript залишається найпоширенішою мовою програмування, що підкреслює його важливість у сучасній веб-розробці. HTML/CSS та Python також демонструють високу популярність, що вказує на їхню значущість у фронтенд-розробці та аналізі даних.

Визначені тенденції відображають поточні потреби індустрії програмного забезпечення та вказують на важливість володіння мовами програмування для сучасних розробників. C# залишається у десятці найкращих мов програмування (27.6%).

Однією з головних переваг C# є баланс між продуктивністю й легкістю розробки, його високорівневі конструкції дозволяють розробникам швидко

реалізовувати складні системи, тоді як оптимізація середовища виконання. Завдяки цьому C# забезпечує зручне середовище для створення як прототипів, так і повноцінних комерційних проєктів.

C# забезпечує потужну підтримку об’єктно-орієнтованого програмування (ООП), яке є ідеальним для створення ігрових механік. Ігрові об’єкти, такі як персонажі, вороги або предмети, природним чином моделюються як об’єкти з властивостями та методами [36], що сприяє кращій організації коду та його повторному використанню.

Використання принципів SOLID та патернів проєктування, таких як «фабрика», «стратегія», «спостерігач», значно полегшується завдяки можливостям C#, що дозволить розробникам створювати модульні механіки, які легко розширюються та тестуються.

На відміну від C++, де програміст самостійно відповідає за керування пам’яттю, в C# процес контролюється автоматичним збиранням сміття (Garbage Collection), що зменшує кількість помилок, пов’язаних із витоками пам’яті, та підвищує стабільність ігор.

Завдяки простому синтаксису, широкій базі готових бібліотек і великій кількості документації, C# забезпечує високу швидкість створення перших робочих прототипів ігрових механік, що особливо важливо в умовах Agile-розробки, де потрібна гнучкість і швидкість адаптації. Ігрова механіка – сукупність конкретних дій, правил і систем взаємодії, через які користувач впливає на ігровий світ і отримує зворотний результат, формуючи безпосередній ігровий досвід.

Одним із прикладів є реалізація системи штучного інтелекту ворогів. За допомогою C# розробник може легко організувати стан-машини поведінки ворога, використовуючи шаблон «стан» (State Pattern). Інший приклад – система інвентаря для RPG-ігор, де об’єкти (зброя, броня, артефакти) моделюються через базові класи і розширюються похідними, що дозволяє ефективно додавати нові типи предметів.

Зазначимо, що Godot Engine – один із найперспективніших інструментів для розробки ігор, починаючи з версії 3.0, рушій надає підтримку мови програмування C#, що стало важливою віхою для розробників, які прагнуть поєднати зручність Godot із потужністю типізованого кодування. Використання C# у Godot відкрило нові можливості для створення складних, масштабованих ігрових механік, зберігаючи при цьому ефективність процесу розробки.

Спочатку основною мовою для скриптування у Godot була GDScript – спеціалізована мова, натхненна Python. Проте додавання підтримки C# через інтеграцію із середовищем .NET (Mono) суттєво розширило функціональні можливості рушія. Інтеграція C# у Godot має такі особливості як: використання Mono-версії рушія для доступу до повноцінного .NET API; можливість використовувати зовнішні бібліотеки .NET і NuGet-пакети; автоматичне прив’язування C# скриптів до ігрових об’єктів (Node); та ін.

Можливості рушія суттєво збільшують спектр застосування Godot як для інді-проєктів.

C# – мова зі строгою типізацією, що дозволяє виявляти помилки ще на етапі компіляції, а не під час виконання програми [36], що важливо для забезпечення стабільності логіки гри, особливо коли йдеться про взаємодію великої кількості об’єктів. GDScript, попри зручність, має динамічну типізацію, що підвищує ризик помилок у великих проєктах. C# дає змогу уникнути цих недоліків і гарантувати вищу якість коду.

Godot за своєю природою побудований на системі вузлів (Node System), де кожен елемент сцени є окремим об’єктом. C# із потужною підтримкою ООП ідеально поєднується з даною моделлю, дозволяючи легко реалізовувати ієрархії об’єктів, успадкування та композицію [38]. Наприклад, базова механіка керування персонажем може бути реалізована як клас Player, що успадковує поведінку від KinematicBody2D, додаючи специфічні властивості, такі як швидкість, здоров’я чи методи взаємодії з оточенням.

C# надає вбудовані механізми для реалізації асинхронного коду через async та await, що спрощує роботу зі складними механіками на кшталт завантаження сцен у фоні, роботи мережевих запитів або обробки тривалих обчислень без заморожування кадру [35]. Для ігрових механік, таких як процедурна генерація рівнів або обробка великої кількості NPC, можливість легкого асинхронного виконання завдань є надзвичайно корисною.

Використання C# у Godot дозволяє розширити функціональність проєкту за рахунок існуючих .NET-бібліотек: систем штучного інтелекту, обробки фізики, мережевих взаємодій тощо. Підрахунок дозволить розробникам зосередитись на розробці ігрового процесу, замість створення базової інфраструктури з нуля. Наприклад, для реалізації складної системи діалогів чи збереження стану гри можна використовувати готові рішення із пакунків NuGet.

За допомогою даної мови програмування створено різноманітні ігрові механізми, які охоплюють усі аспекти ігрового процесу, виділимо приклади механік:

1. *Звукові ефекти та музика*: з допомогою C# можна створити адаптивні звукові системи, які змінюються в залежності від дій гравця або подій у грі, створюючи більш глибокий і захоплюючий ігровий досвід.

2. *Механіки руху персонажів*: використання C# для створення системи управління рухом персонажів, включаючи стрибки, біг, плавання, лазіння та інші дії, також включає не тільки анімацію, а й фізичний рух, враховуючи типи різних поверхонь і перешкод.

3. *Мультиплеєрні механіки*: C# використовується для створення системи мережевої взаємодії між гравцями, включаючи синхронізацію даних, управління серверами, обробку подій і комунікацію в реальному часі.

4. *Системи взаємодії з об'єктами*: за допомогою C# розробляються механізми взаємодії гравця з об'єктами та середовищем, такі як підбір об'єктів, відкриття дверей, механізми активації, рішення головоломок тощо.

5. *Системи камер*: C# дозволяє створювати складні системи управління камерами, включаючи слідкування за персонажами, динамічну зміну кутів огляду, зум і перехід між стандартними режимами камери.

6. *Системи управління ресурсами*: C# дозволяє створити складні системи управління ресурсами в іграх, таких як інвентар, економічні системи, системи крафту або управління базами, які часто зустрічаються в стратегічних і ролевих іграх.

7. *Фізичні механіки*: C# дозволяє розробляти реалістичні фізичні симуляції, такі як гравітація, зіткнення, сила тертя та рух об'єктів. Unity, який використовує C#, має потужний фізичний рух, що дозволяє створювати складні взаємодії між об'єктами.

8. *Штучний інтелект (AI)*: C# широко використовується для створення алгоритмів поведінки NPC (неігрових персонажів), включаючи прийняття рішень, патрулювання, реакцію на дії гравця, стратегічне мислення та інші форми інтелектуальної поведінки (рис. 1.18).

```
public interface IAttackable
{
    void TakeDamage(int amount);
}

public class Enemy : KinematicBody2D, IAttackable
{
    public int Health = 100;

    public void TakeDamage(int amount)
    {
        Health -= amount;
        if (Health <= 0)
            QueueFree(); // Видаляємо об'єкт зі сцени
    }
}
```

Рис. 1.18. Створення системи бойових механік

(джерело: авторська розробка)

У Godot за допомогою C# можна створити систему ближнього бою, в якій кожен персонаж реалізує інтерфейс IAttackable, що визначає метод TakeDamage. Такий підхід дає можливість гнучко змінювати тип ворогів та їхню поведінку без необхідності змінювати основний бойовий код (рис. 1.9).

C# дозволяє зручно організувати систему квестів через шаблони проєктування (наприклад, «команда» або «спостерігач»). Події можуть передаватися між об’єктами за допомогою делегатів, що значно підвищує модульність і розширюваність системи.

Хоча C# має багато переваг, він не позбавлений недоліків. Зокрема, у випадках надто великих проєктів або специфічних вимог до максимальної продуктивності (наприклад, у великих AAA-проєктах) розробники можуть надавати перевагу C++ через його нижчий рівень доступу до ресурсів системи. Також існують обмеження на використання певних можливостей апаратного прискорення через середовище CLR.

При використання C# у Godot виникає необхідність встановлення додаткових компонентів (Mono-версії рушія); трохи довший час компіляції в порівнянні з GDScript; дещо вищі вимоги до ресурсів системи. Однак для більшості проєктів недоліки є незначними і повністю перекриваються перевагами, які C# приносить у розробку.

Підтримка C# у Godot Engine суттєво розширює можливості розробників у створенні складних ігрових механік. Строга типізація, об’єктно-орієнтований підхід, асинхронне програмування та доступ до екосистеми .NET роблять C# одним із найкращих виборів для середніх і великих проєктів. Використання обраної мови програмування забезпечує підвищення надійності коду, швидкість розробки та масштабованість ігрових систем, що є критично важливим у трансформації створення ігор.

C# є однією з найкращих мов програмування для створення ігрових механік завдяки своїй простоті, гнучкості та тісній інтеграції з популярними ігровими рушіями. Високий рівень абстракції, підтримка ООП, безпечне керування пам’яттю і можливість швидкого прототипування роблять його надзвичайно привабливим вибором для інді-розробників і невеликих студій.

Висновок до першого розділу

Проведений у розділі аналіз, присвячений сучасним інструментам створення ігрового контенту, засвідчує, що ігрова розробка перебуває на етапі інтенсивної цифрової трансформації. Вона поступово виходить за межі суто інженерного підходу та набуває міждисциплінарного характеру, поєднуючи програмування, дизайн, моделювання, звукову інженерію, когнітивні технології та елементи штучного інтелекту. У зв’язку з цим зростають вимоги до інструментарію, який визначає не лише ефективність процесу розробки, а й рівень інноваційності та конкурентоспроможності кінцевого продукту.

Встановлено, що домінуючим напрямом розвитку галузі є використання ігрових рушіїв як універсальних технологічних платформ. Такі рушії забезпечують керування графікою, фізикою, анімацією, логікою та взаємодією користувача з ігровим середовищем. Провідні рушії сучасності, зокрема Unity, Unreal Engine та Godot, стали основою більшості ігрових проєктів, оскільки дозволяють реалізовувати як прості двовимірні ігри, так і складні тривимірні інтерактивні середовища. Аналіз підтверджує, що ефективна розробка неможлива в межах одного вузького інструмента, а потребує комплексного середовища, яке об’єднує графіку, скриптування, анімацію, інтерфейс користувача та аудіо.

Окрему увагу приділено поширенню рушіїв із відкритим вихідним кодом, що суттєво вплинуло на демократизацію ігрової розробки. Доступність сучасних технологій дала змогу значно розширити коло розробників, зокрема інді-спільноту та початківців, що, у свою чергу, сприяло появі нових жанрів, форматів і підходів до створення ігор.

У межах розділу ґрунтовно охарактеризовано рушій Godot Engine як перспективну платформу для сучасної ігрової розробки. Проведений аналіз показав, що Godot вирізняється гнучкою архітектурою, підтримкою 2D- та 3D-середовищ, відкритістю, кросплатформенністю, а також наявністю

вбудованих інструментів для анімації, сценно-орієнтованої логіки та підтримки кількох мов програмування. Важливою перевагою рушія є його модульність, передбачуваність виконання коду та орієнтація на оптимізацію, що забезпечує масштабованість і зручність розробки проєктів різної складності.

Аналіз тенденцій популярності Godot підтверджує його динамічний розвиток і поступовий перехід із категорії альтернативних інструментів до сегмента повноцінних професійних рішень. Особливо значущим є застосування рушія в освітньому середовищі, оскільки його доступність та відкритість суттєво знижують поріг входження у сферу геймдеву.

Отже, узагальнення матеріалу дозволяє зробити висновок, що використання сучасних ігрових рушіїв і цифрових інструментів є визначальним чинником ефективної розробки ігрових систем. Застосування Godot Engine у поєднанні зі спеціалізованими засобами графічного моделювання, анімації, аудіообробки та оптимізації формує цілісну технологічну екосистему, яка забезпечує повний цикл створення інтерактивного ігрового продукту та створює методологічну основу для переходу до практичної реалізації програмного проєкту.

РОЗДІЛ 2
РОЗРОБКА ІГРОВОГО ПРОЄКТУ ВИКОРИСТАННЯМ РУШІЯ GODOT ТА C#

2.1 Реалізація ігрової логіки з використанням C#

Інтенсивний розвиток цифрових технологій зумовлює розширення сфер застосування комп’ютерних ігор, що дедалі частіше використовуються не лише як засіб розваги, а й як інструмент навчання, моделювання, професійної підготовки та формування цифрових компетентностей. Зростання вимог до якості ігрового програмного забезпечення, рівня інтерактивності та продуктивності стимулює впровадження сучасних рушіїв розробки, здатних інтегрувати засоби програмування, тривимірного моделювання, анімації та оптимізації обчислювальних процесів у єдине технологічне середовище. Серед таких платформ особливу увагу привертає Godot Engine, що характеризується відкритою архітектурою, кросплатформенністю та широкими можливостями для створення складних інтерактивних систем.

Використання мови програмування C# у поєднанні з рушієм Godot забезпечує реалізацію об’єктно-орієнтованого підходу до побудови ігрової логіки, розроблення масштабованих програмних компонентів і впровадження ефективних алгоритмів керування ресурсами. Гнучкість архітектури рушія та продуктивність C# створюють передумови для формування стабільних, функціонально насичених та оптимізованих ігрових середовищ.

Розвиток сучасних цифрових технологій суттєво розширює можливості програмування інтерактивних систем, серед яких комп’ютерні ігри займають провідне місце як складні програмні комплекси, що поєднують алгоритмічну обробку даних, моделювання поведінкових процесів та засоби взаємодії користувача з віртуальним середовищем.

Ключовим компонентом будь-якої ігрової системи виступає ігрова логіка, що визначає правила функціонування ігрового світу, обробку подій, взаємодію об’єктів, керування станами персонажів та реалізацію сценарних механік. Ігрова логіка – сукупність правил, алгоритмів і програмних сценаріїв, що визначають поведінку всіх елементів гри та умови їхньої взаємодії між собою й із користувачем.

Мова програмування C# є одним із найбільш ефективних інструментів створення ігрової логіки завдяки поєднанню продуктивності, зручності об’єктно-орієнтованого підходу та сучасних засобів асинхронного програмування. Розвинений набір бібліотек, підтримка багатопотоковості, сувора типізація та синтаксична структурованість забезпечують можливість розроблення масштабованих, модульних і підтримуваних програмних систем. У контексті ігрової розробки використання C# дозволяє ефективно реалізовувати поведінкові алгоритми персонажів, системи інвентаря та ресурсів, механіки взаємодії, керування подіями й тригерами, а також модулі адаптації ігрового процесу до дій користувача.

Організація структури програмних модулів у проєкті, розробленому на основі рушія Godot із використанням мови C#, передбачає побудову чіткої ієрархії компонентів та визначення принципів їх взаємодії. Основою архітектури є поділ функціональності на окремі логічні блоки, кожен з яких відповідає за конкретний аспект ігрового процесу: управління ресурсами, обробку інтеракцій, візуальне відображення, внутрішню логіку крафту тощо. Така модульність забезпечує структурованість та спрощує подальшу підтримку проєкту, надаючи можливість розширювати функціонал без зміни базових частин системи.

Для забезпечення узгодженості між окремими елементами застосовано принципи чіткої залежності та мінімізації зв’язності між модулями. Комунікація між системами відбувається через визначені інтерфейси або подієві механізми рушія, що дозволяє зменшити кількість прямої взаємодії та забезпечує можливість заміни або модифікації окремих компонентів без

втручання в інші частини проєкту. Такий підхід сприяє підвищенню масштабованості, а також покращує читабельність і повторне використання коду в межах складного 3D-ігрового середовища:

- абстракція – виділення суттєвих характеристик об’єктів і приховування деталей реалізації;
- інкапсуляція – приховування внутрішнього стану об’єктів і надання керованого інтерфейсу для взаємодії;
- модульність – поділ системи на незалежні компоненти для спрощення розробки та підтримки;
- слабка зв’язність – мінімізація залежностей між модулями для підвищення гнучкості та розширюваності.

Окрему місце у структуризації взаємодії між модулями відіграє сформована інфраструктура керування залежностями, що реалізована засобами мови C#. Вона забезпечує контрольоване створення та використання об’єктів, а також дозволяє централізовано управляти життєвим циклом компонентів, що взаємодіють в межах гри. Завдяки цьому вдалося досягти розмежування відповідальностей між частинами системи, оптимізувати порядок ініціалізації та забезпечити гнучкість у розширенні функціональності під час збільшення масштабів проєкту (рис. 2.1).

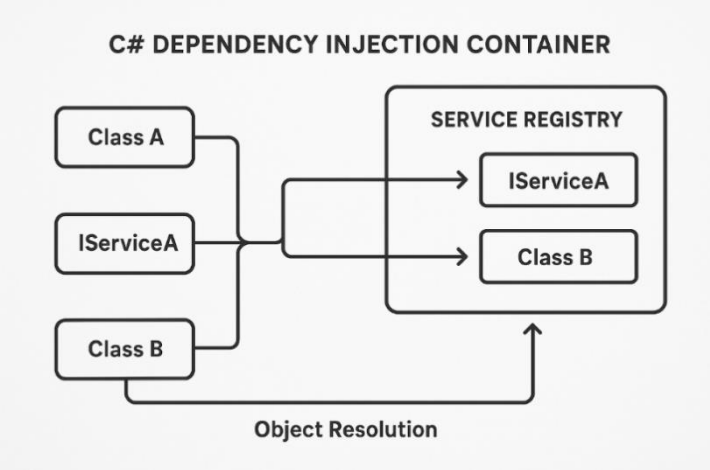

Рис. 2.1. Впровадження залежностей C#

Під час розробки ігрового проєкту на рушії Godot із використанням мови C# особлива увага приділяється механізмам абстракції, які дозволяють

приховати внутрішню реалізацію окремих систем та забезпечити єдиний інтерфейс взаємодії. Абстракція дає змогу розділити логіку гри на незалежні компоненти, кожен з яких виконує чітко визначену функцію: управління об’єктами, обробка подій, системи переробки ресурсів або крафтінгу, що спрощує тестування окремих модулів та дозволяє уникати дублювання коду при впровадженні нових механік.

Розширюваність систем досягається шляхом використання узгоджених контрактів між модулями та застосування принципів об’єктно-орієнтованого програмування, таких як наслідування та поліморфізм, що дозволяє додавати нові функції або змінювати поведінку існуючих компонентів без необхідності модифікувати базові модулі. У 3D-середовищі є важливими зазначені підходи для забезпечення взаємодії гравця з великою кількістю об’єктів, оскільки кожен елемент сцени може бути об’єктом окремого класу з власною логікою, реалізованою через абстрактні базові інтерфейси.

Застосування абстракції та розширюваності у поєднанні з контролем залежностей і модульною організацією дозволяє створити гнучку архітектуру, яка адаптується до змін у ігровому процесі та масштабу проєкту. Таке поєднання сприяє ефективній підтримці ігрових механік, дозволяє інтегрувати нові системи та знижує ризик помилок при розширенні функціоналу. Візуалізація взаємодії абстрактних компонентів із конкретними реалізаціями допомагає наочно демонструвати логіку роботи системи та полегшує планування подальшого розвитку проєкту.

Розробка механізмів інтеракції гравця з об’єктами тривимірного середовища є ключовою складовою функціоналу ігрового проєкту (рис. 2.2), адже передбачає забезпечення зручного та інтуїтивного способу взаємодії користувача з предметами, які знаходяться у просторі бункера. Для цього застосовуються принципи подієвої взаємодії та обробки колізій, що дозволяє визначати момент контакту гравця з конкретним об’єктом і відповідно реагувати на нього у програмній логіці.

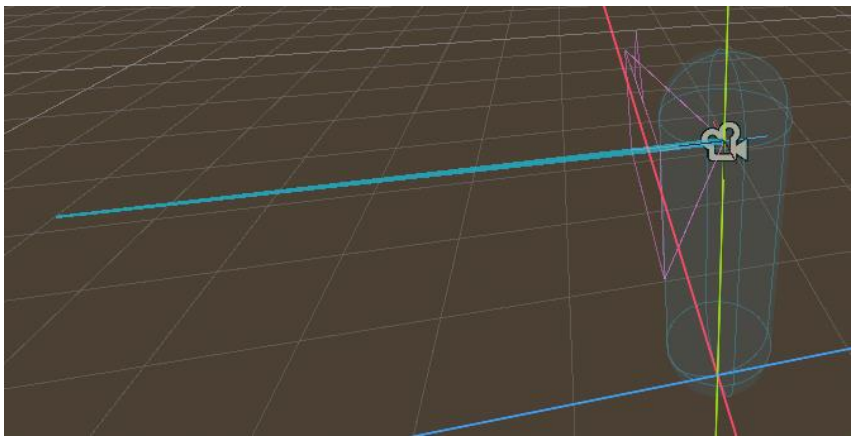

Рис. 2.2.Сцена головного гравця

Кожен інтерактивний елемент має власні властивості та методи, що визначають його поведінку під час взаємодії, що дозволяє реалізувати різні типи об’єктів, такі як ресурси для збору, контейнери для зберігання або станції переробки матеріалів. Завдяки модульній організації системи інтеракцій можна легко додавати нові об’єкти або змінювати їх поведінку, не порушуючи роботу інших компонентів проєкту.

Впровадження механізмів інтеракції включає також відображення стану об’єктів для гравця, що підвищує зручність та наочність ігрового процесу (рис. 2.3). Візуальні підказки, анімації підняття або обробки предметів та відповідні зміни в інвентарі забезпечують зворотний зв’язок, що дозволяє гравцю легко орієнтуватися у просторі бункера та планувати свої дії. Такий підхід сприяє глибині ігрового досвіду та покращує загальну якість взаємодії користувача із середовищем.

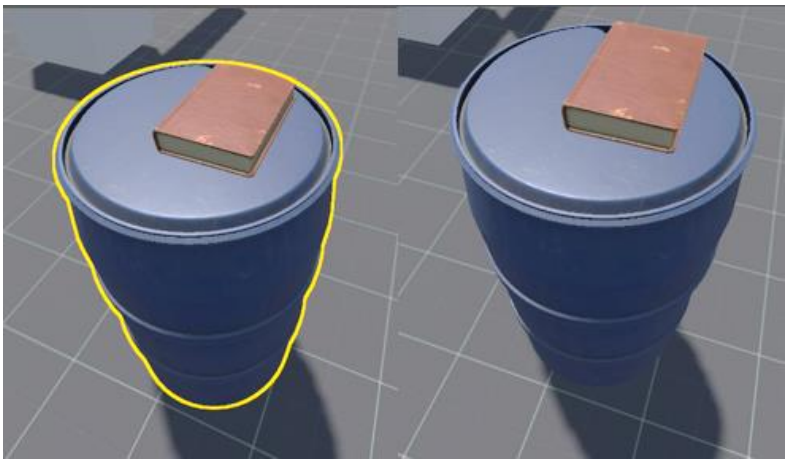

Рис. 2.3. Виділення об’єктів

Система управління інвентарем у тривимірному ігровому середовищі є основним елементом організації ігрового процесу, що забезпечує зберігання, сортування та відображення ресурсів і предметів, які збирає гравець. Вона реалізується за допомогою модульної структури, де кожен об’єкт інвентаря має визначені властивості та методи для взаємодії з іншими компонентами системи. Така організація дозволяє забезпечити динамічне оновлення інвентаря під час збору ресурсів або створення нових предметів, а також спрощує подальшу інтеграцію нових типів об’єктів.

Система управління ресурсами передбачає класифікацію та облік зібраних елементів, що використовуються у процесах переробки та крафту. Кожен ресурс має власні характеристики, які впливають на можливості комбінування та виробництва готових предметів. Завдяки централізованому управлінню ресурсами можна ефективно контролювати їх наявність, відстежувати зміни під час обробки та забезпечити узгодженість у всіх процесах взаємодії з об’єктами бункера (рис. 2.4).

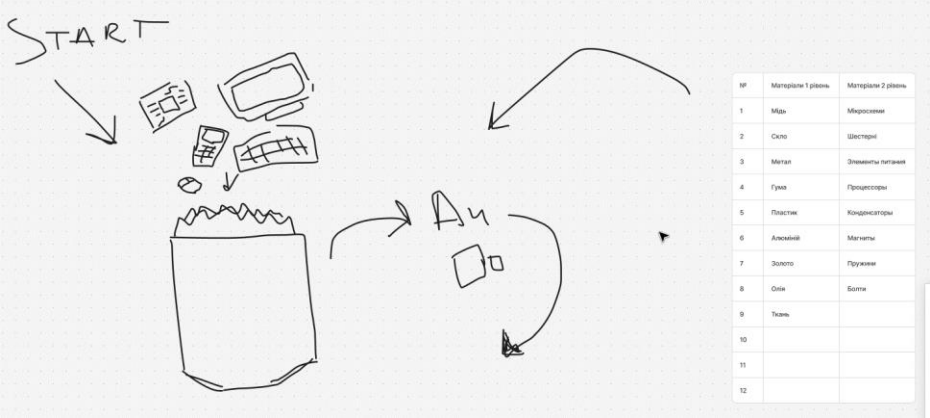

Рис. 2.4. Схема роботи з матеріалами

Процеси переробки реалізуються через інтерактивні об’єкти, такі як верстаки або станції переробки, де зібрані ресурси трансформуються у компоненти для створення готових виробів. Логіка переробки організована таким чином, щоб забезпечувати послідовність операцій і контроль стану матеріалів на кожному етапі. Візуалізація процесу та відповідний зворотний зв’язок дозволяють гравцю оцінювати результати своїх дій і планувати подальші кроки, що підвищує ефективність ігрового досвіду.

Для забезпечення масштабованості та підтримки модульної структури ігрового проєкту була реалізована власна інфраструктура керування залежностями на базі мови C#. Основною метою такої системи є централізоване управління створенням та доступом до об’єктів різних модулів, що дозволяє зменшити прямі залежності між компонентами та підвищує гнучкість архітектури. Інфраструктура забезпечує автоматичну реєстрацію об’єктів, їх ініціалізацію та зв’язування через механізми атрибутів, що визначають, які елементи підлягають реєстрації та ін’єкції.

```
7 references
public partial class ProjectContext : CompositionRoot
{
    [Export, Register] private StageManager stageManager;
    [Export, Register] private InputManager inputManager;

    [DeepRegister] private GameDB gameDB = new();
}
```

Рис. 2.5. Глобальний контекст

Механізм керування залежностями реалізує принципи глибокої реєстрації та розв’язання сервісів, що дозволяє автоматично відслідковувати і підключати залежності всередині складних об’єктів. Система обробляє поля та властивості класів із застосуванням спеціальних атрибутів, таких як RegisterAttribute, InjectAttribute та DeepRegisterAttribute, забезпечуючи належну ін’єкцію сервісів та компонентів на всіх рівнях ієрархії, що дозволяє забезпечити цілісність роботи системи навіть при великій кількості взаємозалежних модулів.

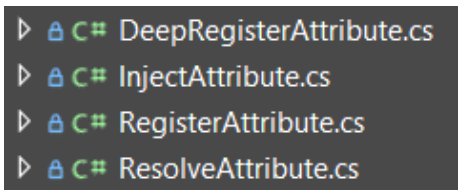

Рис. 2.6. Атрибути впровадження залежностей

Реалізація власного DI-контейнера дозволяє не лише керувати об’єктами всередині одного контексту, але й організувати багаторівневу ієрархію контекстів із можливістю пошуку сервісів у батьківських контейнерах. Така архітектура забезпечує масштабованість проєкту та зручність додавання нових компонентів без модифікації існуючих модулів.

Візуалізація потоків реєстрації та розв’язання залежностей допомагає відстежувати взаємодію між компонентами та оцінювати ефективність організації системи, що особливо важливо для складних ігрових механік у тривимірному середовищі.

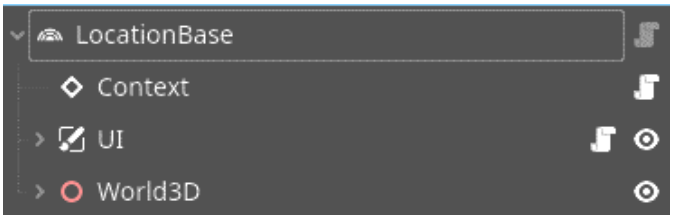

Рис. 2.7. Ієрархія сцени локації

Як результат, поєднання рушія Godot із використанням мови програмування C# створює ефективну технологічну основу для реалізації складної та багаторівневої ігрової логіки. Застосування принципів об’єктно-орієнтованого програмування, зокрема абстракції, інкапсуляції, модульності та слабкої зв’язності, забезпечує чітку структуризацію програмних компонентів і сприяє формуванню гнучкої архітектури, здатної до масштабування та подальшого розширення функціоналу. Організація систем взаємодії через подієві механізми та визначені інтерфейси дозволяє мінімізувати прямі залежності між модулями, підвищити читабельність коду й забезпечити його повторне використання.

Реалізація механізмів інтеракції гравця з елементами тривимірного середовища, формування систем управління інвентарем і ресурсами, а також впровадження логіки переробки матеріалів свідчать про можливість комплексного моделювання ігрових процесів засобами C#. Застосування власної інфраструктури керування залежностями та реалізація DI-контейнера забезпечили централізоване керування життєвим циклом компонентів і підтримку багаторівневої системи сервісів, що сприяє стабільності функціонування ігрового середовища та спрощує впровадження нових механік.

2.2 Проєктування та впровадження ігрових механік

Проєктування ігрових механік становить один із основних етапів створення інтерактивного ігрового середовища, оскільки саме від продуманості їх структури та способів реалізації залежить логічна цілісність ігрового процесу, рівень залучення користувача та функціональна повнота програмного продукту. Проєктування – цілеспрямований творчо-інженерний процес розроблення моделі об’єкта, системи або процесу з визначенням їх структури, функцій і способів практичної реалізації.

Ігрові механіки охоплюють систему правил, алгоритмів і процедур, що визначають поведінку ігрових об’єктів, взаємодію гравця із середовищем, послідовність подій та умови досягнення ігрових цілей. Коректне проєктування таких механік потребує чіткої структуризації логіки гри, визначення сценарних взаємозв’язків і оптимального розподілу функціональних ролей між програмними компонентами.

Варто виокремити, що «ігрові механіки – сукупність правил, алгоритмів та процедур, що визначають взаємодію гравця з ігровим середовищем і формують логіку перебігу ігрового процесу» [30]. Та основними елементами ігрової механіки є формування ігрових механік бере початок із раннього етапу розвитку електронних ігор, що характеризувався використанням простих візуальних елементів і базових моделей руху об’єктів на екрані. Поява аркадних ігор у середині XX століття заклала первинні засади ігрової взаємодії, серед яких домінували елементарні принципи керування, накопичення очок та поступове підвищення рівня складності ігрових завдань. Означені елементи стали основою формування базових структур ігрових систем та визначили подальший розвиток ігрового програмування. Подальший технологічний прогрес, пов’язаний із поширенням персональних комп’ютерів і гральних консолей, сприяв істотному ускладненню архітектури ігрових механік та розширенню можливостей інтерактивної взаємодії користувача з цифровим середовищем.

Упровадження багатокомпонентних систем керування, механізмів винагород і штрафів, адаптивних рівнів складності та нелінійних сюжетних моделей надало розробникам значно ширші можливості для створення варіативного ігрового досвіду.

У сучасному розумінні ігрові механіки являють собою не лише формалізований набір правил функціонування гри, а комплексну систему організації взаємодії гравця з віртуальним середовищем, що поєднує алгоритмічні рішення, поведінкові моделі та інтерактивні сценарії. Вивчення основних характеристик ігрових механік дозволяє визначити принципи побудови ефективного ігрового процесу, спрямованого на формування змістовного та емоційно насиченого користувацького досвіду.

Тому, використання рушія Godot у поєднанні з мовою програмування C# забезпечує можливість реалізації ігрових механік на основі модульної архітектури, об’єктно-орієнтованого підходу та подієвої моделі взаємодії. Застосування зазначених інструментів дозволяє створювати гнучкі та масштабовані системи взаємодії, інтегрувати поведінкові алгоритми, механізми обробки подій, керування станами та ресурсами, а також забезпечувати адаптацію ігрового процесу відповідно до дій користувача.

Подальший виклад спрямовано на розкриття практичних аспектів розроблення та впровадження базових ігрових механік, що формують основу функціонування створюваного ігрового проєкту.

Механізм виявлення та підбору предметів у тривимірному просторі є ключовим елементом взаємодії гравця з ігровим середовищем. Для реалізації цього механізму було використано технологію RayCast3D, яка дозволяє визначати точку перетину променя, випущеного від камери гравця, з об’єктами сцени (рис. 2.8). Такий підхід забезпечує точне визначення об’єктів, доступних для підбору або взаємодії, і дозволяє гнучко налаштовувати зону дії променя та умови виявлення, враховуючи відстань і напрямок глядача.

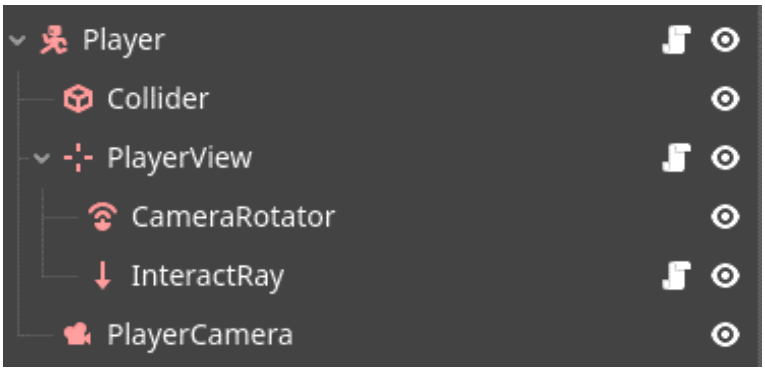

Рис. 2.8. Ієрархія сцени гравця

З метою підвищення ефективності візуального орієнтування користувача у процесі взаємодії з ігровими об’єктами впроваджено систему їхнього виділення шляхом застосування технології Outline, реалізованої на основі SubViewportContainer та окремої камери.

Використання графічного 2D Outline Shader для формування контурного накладання на тривимірні моделі забезпечує візуальне акцентування інтерактивних елементів сцени, що сприяє підвищенню рівня наочності ігрового середовища та полегшує сприйняття доступних для взаємодії об’єктів. Зазначений підхід дозволяє оперативно ідентифікувати предмети, придатні для підбору або подальшої обробки, забезпечуючи інтуїтивно зрозумілу навігацію в межах ігрового простору.

Механізм обробки предметів інтегрується з системою інвентарю та процесами переробки, що дозволяє після підбору відразу додавати об’єкт до ресурсів для крафту. Логіка інтеракції враховує умови підбору, стан об’єкта та його подальшу трансформацію в системі переробки. Таке поєднання точного виявлення та візуальної підказки забезпечує плавну, зрозумілу та ефективну взаємодію гравця з тривимірним середовищем бункера (рис. 2.9).

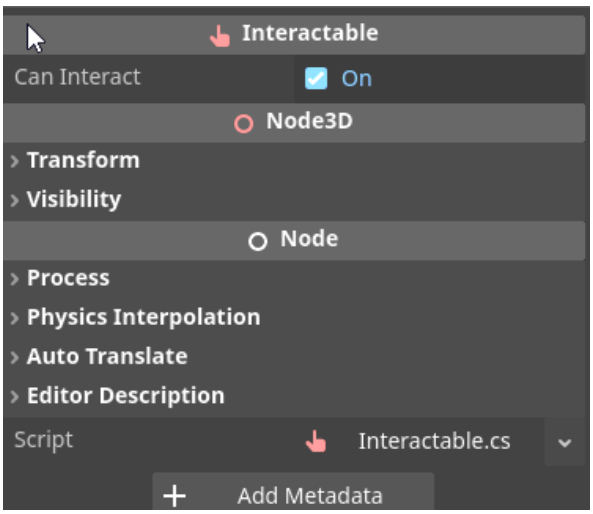

Рис. 2.9. Вузол взаємодії

Система перетворення зібраних об’єктів на компоненти забезпечує логіку трансформації ресурсів, знайдених гравцем у тривимірному середовищі, у матеріали для подальшого крафту. Кожен зібраний предмет має набір властивостей, які визначають, які компоненти можна отримати після обробки. Система реалізована через окремий модуль, який обробляє взаємодії між об’єктами інвентаря та станціями переробки, визначає результати обробки та оновлює стан ресурсів у системі інвентарю.

Процес перетворення організований таким чином, щоб забезпечити послідовність операцій та контроль над станом об’єктів. Кожен компонент отримує атрибути вихідного предмета, а логіка перетворення враховує тип, якість і кількість ресурсів. Такий підхід дозволяє реалізувати багаторівневу систему крафту, де одні компоненти можуть бути об’єднані для створення більш складних виробів, забезпечуючи глибину ігрового процесу та інтерактивність в межах тривимірного середовища.

Система інтегрована з механіками інвентарю та процесами виконання замовлень, що дозволяє після перетворення ресурсів безпосередньо використовувати їх для виготовлення предметів на замовлення. Візуальні підказки та анімації процесу обробки підвищують наочність та інтуїтивність взаємодії, дозволяючи гравцю оцінювати результати дій і планувати наступні кроки. Такий підхід забезпечує організовану, гнучку та розширювану структуру системи крафту.

Механіка комбінування компонентів у готові вироби реалізує логіку крафту, яка дозволяє гравцю створювати предмети на основі зібраних і перероблених ресурсів. Кожен компонент має набір характеристик та вимог для комбінування, що визначає його роль у виробі. Система контролює відповідність компонентів рецепту, облік ресурсів та результативність операції, забезпечуючи коректність створених виробів та відсутність помилок у процесі крафту (рис. 2.10).

Рис. 2.10. Об’єкт підстанції

Для підвищення гнучкості та масштабованості механіки комбінування застосовується модульний підхід, де рецепти та типи виробів описані окремими об’єктами з можливістю динамічного додавання нових комбінацій, що дозволяє інтегрувати нові види виробів без зміни базової логіки системи та забезпечує легке розширення функціоналу. Крім того, система взаємодіє з інвентарем і переробкою ресурсів, що забезпечує цілісність процесу від збору об’єктів до виготовлення готового виробу.

Візуалізація процесу комбінування та відповідні підказки для гравця забезпечують наочність та інтуїтивність крафту. Анімації та графічне відображення результату операції дозволяють гравцю оцінити ефективність створеного виробу і планувати подальші дії. Такий підхід забезпечує

організовану, контрольовану та розширювану систему крафту, що є важливою складовою ігрового процесу у тривимірному середовищі бункера.

Внутрішня система отримання та виконання замовлень організовує взаємодію гравця із завданнями, які формуються для виготовлення предметів на основі перероблених компонентів. Кожне замовлення містить інформацію про необхідні компоненти, кількість та пріоритет виконання. Система контролює стан замовлень, відслідковує їх виконання та забезпечує оновлення даних у реальному часі, дозволяючи гравцю ефективно планувати процес крафту і розподіляти ресурси.

Зберігання та організація замовлень реалізована через внутрішню структуру даних, що дозволяє швидко додавати, видаляти або оновлювати замовлення без втрати інформації. Кожен об’єкт замовлення містить посилання на необхідні ресурси та компоненти, інтегруючись з системами інвентарю та переробки. Такий підхід забезпечує централізований контроль над усіма етапами виробничого циклу та дозволяє реалізувати масштабовану логіку виконання завдань у тривимірному середовищі бункера (рис. 2.11).

Рис. 2.11. Об’єкт робочого місця

Виконання замовлень поєднує логіку контролю компонентів, переробки ресурсів та комбінування готових виробів. Система забезпечує відстеження прогресу та інформування гравця про результати виконання, включаючи візуальні підказки та анімації готових виробів. Такий

комплексний підхід до управління замовленнями дозволяє підтримувати цілісність ігрового процесу, покращує взаємодію користувача із середовищем та підвищує ефективність ігрової механіки.

Інтерактивне обладнання бункера виступає ключовим елементом ігрового процесу, забезпечуючи гравцю можливість перетворювати ресурси, комбінувати компоненти та виконувати замовлення. Станції переробки, робочі столи та контейнери інтегруються з системами інвентарю, механіки підбору та крафту, що дозволяє організувати логічний і послідовний цикл обробки ресурсів. Кожний об’єкт обладнання оснащується власними точками взаємодії та логікою обробки, що дозволяє гравцю ефективно виконувати дії у тривимірному середовищі бункера.

Для підвищення наочності та зрозумілості процесів використовується система підсвітки та візуальних ефектів, що відображає активні точки взаємодії та стан обладнання. Наприклад, при підході до станції переробки активується підсвітка, яка сигналізує про можливість обробки компонентів, а на робочих столах відображається стан процесу крафту. Такий підхід забезпечує інтуїтивність взаємодії та дозволяє гравцю швидко орієнтуватися у просторі бункера.

Інтеграція обладнання з усіма системами ігрового процесу забезпечує цілісність і логічну послідовність дій. Контейнери зберігають ресурси, робочі столи виконують функцію проміжної обробки, а станції переробки трансформують матеріали у компоненти для виготовлення готових виробів. Такий комплексний підхід дозволяє реалізувати організовану, масштабовану та гнучку систему ігрових механік, яка забезпечує глибину та інтерактивність процесу для гравця. Отже, основні функції інтерактивного обладнання:

- зберігання ресурсів може забезпечення накопичення та доступу до зібраних об’єктів;
- інтеграція з інвентарем, автоматичне взаємозв’язування з системою гравця для управління ресурсами

- комбінування та створення готових виробів із компонентів відповідно до замовлень;
- переробка та перетворення ресурсів на компоненти для подальшого використання;
- проміжна обробка та підготовка матеріалів до переробки або комбінування.

Таким чином, інтеграція інтерактивного обладнання з усіма структурними компонентами ігрового процесу забезпечує системну узгодженість функціонування ігрового середовища та логічну послідовність виконання користувацьких дій. Взаємодія контейнерів зберігання, робочих столів і станцій переробки утворює єдиний технологічний цикл обробки ресурсів – від накопичення та підготовки матеріалів до створення готових виробів.

Реалізація автоматизованого зв’язку з системою інвентарю гравця, підтримка процедур комбінування та трансформації ресурсів, а також механізмів проміжної обробки сприяють формуванню цілісної багаторівневої системи ігрових механік. Подібний підхід забезпечує масштабованість архітектури, гнучкість розширення функціоналу й підвищення рівня залученості користувача, що позитивно впливає на інтерактивну глибину та загальну якість ігрового процесу.

Узагальнення матеріалів підрозділу засвідчує, що проєктування та практична реалізація ігрових механік з використанням рушія Godot і мови програмування C# забезпечують створення цілісної, гнучкої й масштабованої системи інтерактивної взаємодії в межах тривимірного ігрового середовища. Реалізовані механізми виявлення та підбору об’єктів на основі технології RayCast3D, впровадження системи візуального акцентування інтерактивних елементів через Outline Shader, інтеграція процесів інвентаря та переробки ресурсів формують зрозумілий і послідовний алгоритм взаємодії користувача з ігровими об’єктами. Системи трансформації ресурсів, крафту та комбінування компонентів у готові вироби забезпечують багаторівневу

логіку виробничих процесів та підвищують змістову насиченість ігрового процесу.

Організація механік виконання замовлень і взаємодії з інтерактивним обладнанням бункера дозволила реалізувати єдиний технологічний цикл обробки ресурсів – від збору та зберігання матеріалів до створення фінальних виробів відповідно до ігрових завдань. Модульний підхід до проєктування систем крафту та управління ресурсами забезпечив можливість динамічного розширення функціоналу без втручання у базову архітектуру програмного продукту. Інтеграція візуальних підказок і анімацій сприяла підвищенню наочності та інтуїтивності ігрової взаємодії, що позитивно вплинуло на рівень залученості користувача.

Отримані результати підтверджують ефективність обраних технологічних і проєктних рішень у побудові ігрових механік засобами C# у середовищі Godot. Сформована система забезпечує структурованість програмної логіки, масштабованість архітектури та стабільну інтерактивну взаємодію у тривимірному середовищі, що створює надійне підґрунтя для подальшого модифікування та удосконалення ігрового проєкту.

2.3. Оптимізація продуктивності та тестування

Забезпечення стабільної продуктивності програмного забезпечення є однією з ключових умов успішної реалізації інтерактивних ігрових систем, оскільки від рівня швидкодії, коректності роботи алгоритмів та відсутності технічних збоїв залежить якість користувацького досвіду й функціональна надійність програмного продукту. Зростання складності тривимірних ігрових середовищ, обсягів графічних ресурсів та кількості обчислювальних процесів зумовлює необхідність застосування спеціалізованих методів оптимізації та системного тестування, спрямованих на зниження навантаження на апаратні ресурси та забезпечення стабільної частоти кадрів.

Оптимізація продуктивності у контексті розробки ігор передбачає комплексний аналіз ефективності використання процесорних, графічних і пам’яттєвих ресурсів, удосконалення алгоритмів обробки даних, адаптацію архітектури програмних компонентів та раціональне керування потоками виконання. «Оптимізація – це сукупність процесів, спрямованих на модернізацію та поліпшення існуючих механізмів досягнення бажаного результату» [19]. Водночас тестування забезпечує виявлення помилок логіки, критичних збоїв та потенційних «вузьких місць» продуктивності шляхом систематичної перевірки функціональності, стрес-навантаження і стабільності роботи програмного середовища (рис. 2.12).

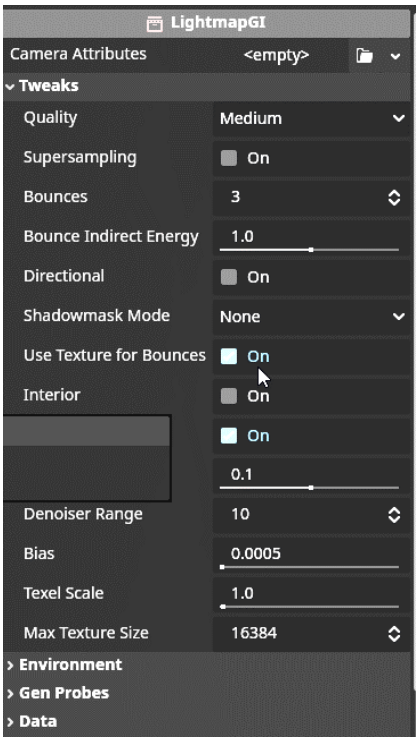

Рис. 2.12. Вузол запеченого світла

Поєднання оптимізаційних методів із різнотипним тестуванням створює необхідні умови для формування надійного, масштабованого та високопродуктивного ігрового продукту. Подальший виклад матеріалу зосереджено на аналізі підходів до підвищення ефективності роботи ігрових механік у тривимірному середовищі та апробації засобів контролю стабільності функціонування програмної системи.

Оптимізація освітлення є важливим елементом забезпечення високої продуктивності тривимірного проєкту, особливо в умовах обмежених апаратних ресурсів. Використання LightmapGI дозволяє попередньо обчислювати глобальне освітлення сцени та зберігати його у текстурах lightmap, що значно знижує навантаження на процесор та графічний рушій під час виконання гри. Такий підхід забезпечує реалістичне та стабільне освітлення без необхідності динамічних обчислень на кожному кадрі.

Додатково застосування Lightmask дозволяє керувати впливом джерел світла на конкретні об’єкти, обмежуючи обчислення лише тими елементами, які дійсно освітлюються, що зменшує кількість непотрібних обчислень та оптимізує використання ресурсів, особливо у складних багаторівневих сценах бункера з великою кількістю інтерактивних об’єктів. Комбінація LightmapGI та Lightmask забезпечує збалансовану продуктивність та якість візуалізації, що важливо для підтримки плавного ігрового процесу.

Оптимізація освітлення інтегрується з іншими методами покращення продуктивності, такими як LOD та Occluder3D, для досягнення максимальної ефективності. Вона дозволяє гравцю отримати високоякісну візуалізацію без падіння частоти кадрів, що підвищує комфорт та занурення у ігрове середовище. Візуальні підказки та коректно налаштоване освітлення також сприяють легшому орієнтуванню у просторі бункера та підвищують інтерактивність ігрового процесу.

Одним із основних методів оптимізації продуктивності в тривимірних сценах є використання Occluder3D, що дозволяє зменшити обчислення для об’єктів, які знаходяться поза зоною видимості гравця. Система Occluder3D визначає геометрію об’єктів, що перекривають інші елементи сцени, та блокує рендеринг прихованих об’єктів, тим самим знижуючи навантаження на графічний процесор, що особливо ефективно у складних сценах бункера з великою кількістю контейнерів, станцій переробки та робочих столів (рис. 2.13).

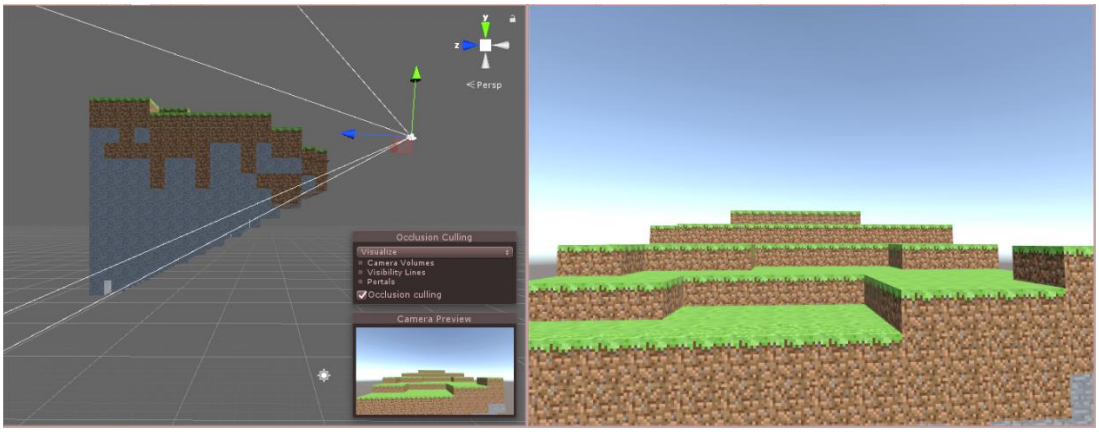

Рис. 2.13. Відсікання геометрії

Використання Occluder3D інтегрується із системами інвентарю та механіки взаємодії з об’єктами, забезпечуючи, що приховані елементи не обчислюються до моменту появи в полі зору гравця. Такий підхід дозволяє підтримувати високу частоту кадрів і стабільність продуктивності навіть у великих тривимірних приміщеннях, де одночасно розташовані численні інтерактивні об’єкти.

Візуальна перевірка та налаштування параметрів Occluder3D дозволяє оптимізувати ефективність рендерингу без втрати якості відображення об’єктів. У поєднанні з іншими методами оптимізації, такими як LOD та LightmapGI, застосування Occluder3D забезпечує комплексний підхід до підвищення продуктивності, дозволяючи створювати реалістичні та інтерактивні сцени з плавним ігровим процесом. Можемо, виділити переваги використання Occluder3D:

- зменшення рендерингу прихованих об’єктів – пропуск об’єктів, що не видно гравцеві;
- інтеграція з LOD та LightmapGI – забезпечення комплексної оптимізації продуктивності сцени;
- підвищення FPS – зменшення навантаження на графічний процесор.

Використання моделей зі зниженим рівнем деталізації (LOD) є важливою складовою оптимізації тривимірного середовища. LOD дозволяє зменшувати складність геометрії та текстур для об’єктів, що знаходяться на віддаленій відстані від гравця, зберігаючи при цьому візуальну цілісність

сцени. У розроблюваному проєкті застосовується комбінований підхід: частина моделей має вже готові LOD як окремі Mesh із налаштуванням visibility distance, тоді як інші використовують автогенерацію LOD у Godot для автоматичного створення спрощених варіантів моделей (рис. 2.14).

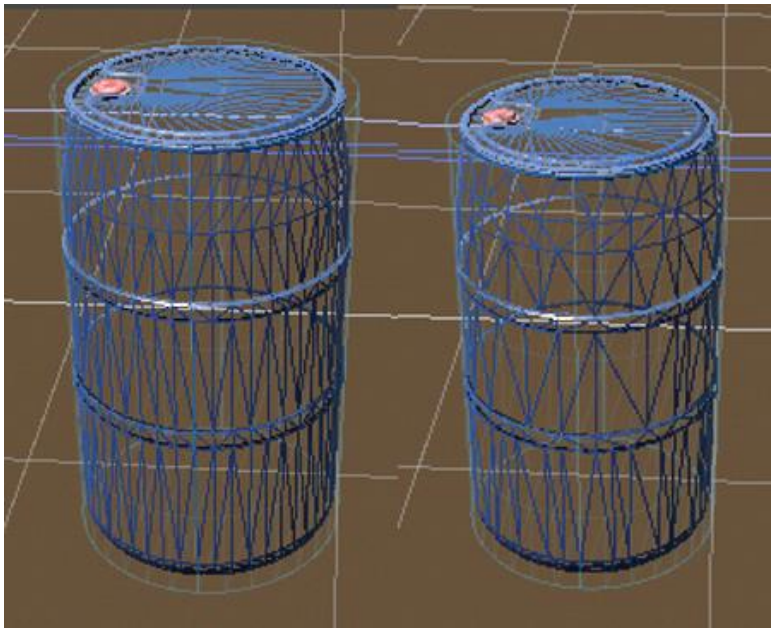

Рис. 2.14. Рівень деталізації

Застосування LOD дозволяє оптимізувати рендеринг великої кількості об’єктів у тривимірному середовищі бункера без помітної втрати якості. Для кожного об’єкта встановлюються відстані, при яких активуються відповідні рівні деталізації, що забезпечує плавну зміну складності моделей та зменшує навантаження на графічний процесор. Комбінований підхід дозволяє максимально ефективно використовувати ресурси: готові LOD забезпечують точну контрольовану якість, а автогенерація дозволяє швидко створювати спрощені моделі для нових або динамічно доданих об’єктів.

Інтеграція LOD з іншими методами оптимізації, такими як LightmapGI та Occluder3D, забезпечує комплексне підвищення продуктивності сцени. Використання LOD не лише знижує кількість полігонів для рендерингу, але й дозволяє зменшити використання відеопам’яті та обчислювальних ресурсів, підтримуючи стабільний FPS та плавність ігрового процесу. Такий підхід дозволяє поєднувати візуальну якість та ефективність рендерингу навіть у великих і деталізованих сценах бункера.

Оптимізація програмних модулів C# здійснюється на основі результатів профілювання продуктивності, що дозволяє визначати «вузькі місця» у виконанні ігрових систем. Використання інструментів профілювання, таких як Godot Profiler, дозволяє аналізувати час виконання окремих методів, обсяг споживаної пам’яті та частоту викликів функцій. На основі отриманих даних виконується рефакторинг коду, оптимізація алгоритмів та перерозподіл обчислень між кадрами для підвищення ефективності (рис. 2.15).

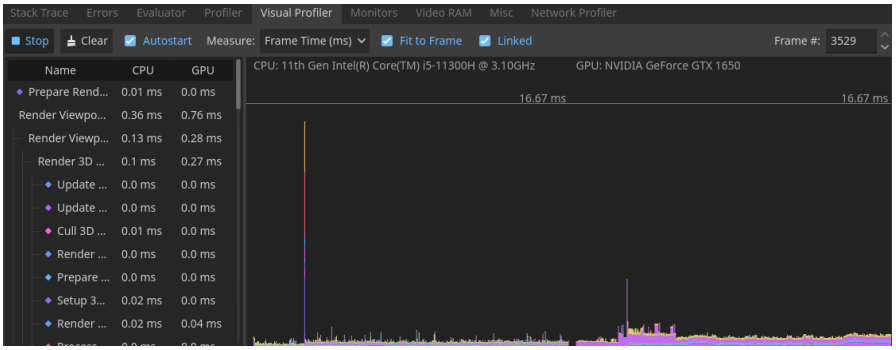

Рис. 2.15. Діагностика продуктивності

Особлива увага приділяється модулям, що взаємодіють із системами інвентарю, переробки ресурсів та управління замовленнями, оскільки вони здійснюють часті обчислення під час ігрового процесу. Оптимізація включає використання кешування результатів, мінімізацію повторних викликів складних методів та застосування ефективних структур даних. Також аналізується використання дженеріків, інтерфейсів та власного DI-контейнера для забезпечення масштабованості, збереження швидкодії та гнучкості модульної структури. До методів оптимізації можемо віднести:

- ефективні структури даних – використання оптимальних структур для швидкого доступу та обробки інформації;
- кешування – збереження результатів обчислень для повторного використання без повторного виконання;
- мінімізація повторних викликів – зменшення кількості викликів ресурсомістких методів;

- оптимізація DI – покращення продуктивності власного контейнера керування залежностями та зменшення накладних витрат на ін’єкції.

Рефакторинг та оптимізація програмних модулів забезпечують стабільну частоту кадрів та зниження споживання пам’яті, що особливо важливо для тривимірного середовища бункера з великою кількістю інтерактивних об’єктів. Комплексний підхід до профілювання та оптимізації дозволяє поєднати високу продуктивність із розширюваною архітектурою систем, забезпечуючи надійність і ефективність ігрового процесу.

Комплексне тестування інтеракцій та функціональних систем гри передбачає перевірку коректності роботи всіх механік взаємодії гравця з тривимірним середовищем бункера. Тестування охоплює систему підбору та обробки об’єктів, управління інвентарем, переробку ресурсів, комбінування компонентів та виконання замовлень. Особлива увага приділяється перевірці взаємодії між підсистемами, щоб забезпечити цілісність процесу та відсутність конфліктів у логіці гри.

Для підвищення ефективності тестування застосовується поєднання ручного та автоматизованого підходів. Ручне тестування дозволяє перевіряти інтуїтивність взаємодії гравця та наочність візуальних підказок, тоді як автоматизовані скрипти забезпечують регулярну перевірку алгоритмів управління інвентарем, крафту та виконання замовлень. Таке поєднання методів дозволяє виявляти помилки на ранніх етапах і гарантувати стабільність роботи систем при зміні конфігурацій та додаванні нових об’єктів.

Результати тестування документуються у вигляді звітів про помилки, графіків продуктивності та статистики стабільності FPS, що дозволяє оцінити якість реалізації інтерактивних механік та ефективність оптимізаційних рішень. Комплексний підхід до тестування забезпечує надійність ігрового процесу, контроль взаємодії всіх підсистем та високу якість кінцевого продукту.

Узагальнення результатів підрозділу дає підстави стверджувати, що застосований комплексний підхід до оптимізації продуктивності й тестування дозволив забезпечити стабільне функціонування розроблюваного ігрового проєкту в умовах високої обчислювальної складності тривимірного середовища. Використання технологій LightmapGI та Lightmask забезпечило ефективну оптимізацію освітлення шляхом попереднього обчислення глобального освітлення та локалізації впливу джерел світла, що дало змогу суттєво зменшити навантаження на графічний рушій без втрати якості візуалізації.

Інтеграція методів Occluder3D сприяла скороченню обсягів рендерингу шляхом усунення обробки прихованих об’єктів, а застосування моделей зі змінним рівнем деталізації (LOD) дозволило оптимізувати геометричну складність сцен залежно від відстані до гравця, забезпечуючи стабільну частоту кадрів і зниження споживання відеопам’яті.

Профілювання програмних модулів C# та подальший рефакторинг коду дозволили ідентифікувати критичні «вузькі місця» у функціонуванні систем інвентарю, переробки ресурсів і керування замовленнями, що дало змогу оптимізувати алгоритми шляхом кешування результатів обчислень, мінімізації повторних викликів ресурсоємних методів та застосування ефективних структур даних. Удосконалення механізмів роботи власного DI-контейнера сприяло збереженню масштабованості архітектури за одночасного зменшення накладних витрат на керування залежностями.

Комплексне тестування інтерактивних механік шляхом поєднання ручних та автоматизованих методів забезпечило перевірку коректності взаємодії між усіма підсистемами гри, стабільності логіки ігрового процесу та відповідності візуальних і функціональних компонентів визначеним вимогам. Документування результатів тестування у вигляді звітів і показників продуктивності дозволило об’єктивно оцінити ефективність упроваджених оптимізаційних рішень.

В результаті є змога стверджувати, що системне поєднання методів оптимізації та тестування створює надійну технологічну основу для формування високопродуктивного, масштабованого та стабільного ігрового програмного продукту.

2.4 Практичні аспекти використання розробленого проєкту

Результати, отримані в процесі виконання дослідження, мають практичну цінність і можуть бути впроваджені у діяльність компанії EXREALITE – студії, що спеціалізується на розробленні інтерактивних цифрових продуктів, зокрема ігор, VR/AR-рішень та 3D-візуалізацій (https://exrealite.com/). Автор роботи на даний час працює у компанії на посаді головного C# програміста. Тематика даного дослідження, пов’язана з оптимізацією продуктивності ігрових проєктів із використанням рушія Godot Engine та мови програмування C#, безпосередньо відповідає напряму діяльності компанії, орієнтованому на створення високоякісних інтерактивних середовищ із підвищеними вимогами до стабільності, швидкодії та масштабованості програмних рішень.

У межах дослідження були розроблені та апробовані підходи до побудови модульної архітектури ігрових систем, оптимізації логіки взаємодії об’єктів, управління ресурсами та зниження навантаження на графічну підсистему. Зазначені підходи можуть бути використані в робочих процесах EXREALITE під час створення тривимірних ігрових сцен, інтерактивних симуляцій та віртуальних середовищ, де важливу роль відіграє ефективне використання апаратних ресурсів, особливо у VR/AR-застосунках.

Практичну цінність для компанії становлять реалізовані в межах роботи методи оптимізації рендерингу, зокрема застосування baked lighting, систем оклюзії та рівнів деталізації (LOD), а також результати профілювання ігрових сцен. Застосування цих підходів дозволяє зменшити навантаження на GPU, підвищити стабільність FPS і забезпечити плавність взаємодії

користувача з цифровим середовищем, що є критично важливим для продуктів, орієнтованих на реальний час та імерсивний досвід.

Окрему практичну значущість мають розроблені архітектурні рішення щодо організації ігрової логіки на основі C#, включно з використанням принципів об’єктно-орієнтованого програмування, модульності та керування залежностями. Запропонована архітектурна модель може бути використана в EXREALITE як шаблон для побудови масштабованих і підтримуваних проєктів, що спрощує подальше розширення функціоналу, командну розробку та технічний супровід програмних продуктів.

Отримані результати також можуть бути застосовані в практичній діяльності компанії, зокрема під час адаптації нових фахівців, стажерів або внутрішнього навчання, як приклад сучасного підходу до проєктування та оптимізації ігрових систем. Таким чином, результати кваліфікаційної роботи можуть бути інтегровані в реальні виробничі процеси EXREALITE, сприяючи підвищенню ефективності розробки, якості кінцевого продукту та конкурентоспроможності компанії на ринку інтерактивних цифрових рішень.

Висновки до другого розділу

В процесі розробки ігрового проєкту було продемонстровано ефективність поєднання рушія Godot та мови програмування C# для створення повноцінного інтерактивного середовища. Ключовими результатами стали організація модульної структури програмних компонентів із забезпеченням чіткої взаємодії між ними, впровадження механізмів абстракції та розширюваності для підтримки масштабованості систем, а також реалізація інтеракцій гравця з тривимірними об’єктами та комплексних процесів управління інвентарем і ресурсами. Впроваджена система власного керування залежностями (DI-контейнер) дозволила централізовано контролювати створення і зв’язування сервісів, що

забезпечило ефективне повторне використання модулів і зниження ризику дублікацій логіки.

Застосування інструментів оптимізації продуктивності, таких як LightmapGI, Lightmask, Occluder3D та LOD, дозволило забезпечити високу частоту кадрів і стабільність роботи тривимірної сцени з великою кількістю інтерактивних об’єктів. Профілювання модулів C# сприяло ідентифікації «вузьких місць» та оптимізації алгоритмів управління інвентарем, переробки ресурсів і комбінування компонентів. Впровадження комплексного тестування інтеракцій і функціональних систем забезпечило контроль цілісності та надійності всіх підсистем, що дозволило гарантувати коректність виконання логіки гри та стабільність взаємодії між компонентами.

Таким чином, розробка проєкту демонструє практичну реалізацію принципів об’єктно-орієнтованого програмування, модульності та масштабованості у контексті комп’ютерних наук. Інтеграція технічних рішень для оптимізації продуктивності та забезпечення стабільної роботи всіх систем підтверджує ефективність використання сучасних методів розробки ігрового програмного забезпечення. Впроваджені підходи формують основу для подальшого розширення функціоналу та вдосконалення ігрових механік з урахуванням вимог до інтерактивності та ресурсної ефективності.

ЗАГАЛЬНІ ВИСНОВКИ

В межах виконання кваліфікаційної роботи було здійснено комплексне дослідження сучасних підходів до розроблення ігрових проєктів із використанням рушія Godot Engine та мови програмування C#. Основну увагу зосереджено на поєднанні теоретичного аналізу тенденцій розвитку розробки ігор з практичною реалізацією власного інтерактивного тривимірного проєкту, що дозволило підтвердити доцільність обраної технологічної платформи та ефективність застосованих методів програмування і оптимізації.

У процесі виконання кваліфікаційної роботи було послідовно реалізовано поставлену мету та виконано всі визначені завдання, що дало змогу комплексно дослідити особливості розроблення та оптимізації ігрових проєктів із використанням рушія Godot Engine та мови програмування C#. Поєднання теоретичного аналізу з практичною реалізацією власного тривимірного інтерактивного проєкту дозволило підтвердити доцільність обраної технологічної платформи та ефективність застосованих архітектурних і програмних рішень.

Виконання першого завдання – аналіз сучасного стану розвитку ігрової індустрії та інструментів створення цифрових ігор – дозволило систематизувати основні тенденції розвитку геймдеву, зокрема зростання ролі кросплатформенних рушіїв, поширення відкритих програмних рішень, розширення освітнього потенціалу ігрових технологій та підвищення вимог до інтерактивності й продуктивності цифрових середовищ. Отримані результати створили теоретичне підґрунтя для обґрунтованого вибору інструментарію практичної частини дослідження.

У межах виконання другого завдання – дослідження функціональних можливостей рушія Godot Engine – здійснено аналіз його архітектури, інструментів та принципів побудови ігрових середовищ. Доведено, що Godot

є конкурентоспроможною універсальною платформою, яка поєднує відкриту архітектуру, підтримку 2D і 3D проєктів, гнучку систему сценарного програмування та повну сумісність із мовою C#. Це обґрунтувало доцільність використання рушія для реалізації масштабованого ігрового проєкту.

Виконуючи третє завдання – визначення методичних засад застосування мови програмування C# у створенні інтерактивних ігрових систем, було розглянуто принципи об’єктно-орієнтованого програмування, подієву модель, модульність і використання шаблонів проєктування. У результаті встановлено, що застосування C# у середовищі Godot дозволяє формувати чітку програмну архітектуру з розподілом відповідальностей між компонентами, що є критично важливим для складних тривимірних ігрових середовищ.

У рамках виконання четвертого завдання – розроблення власного ігрового проєкту – було спроєктовано та реалізовано комплексну архітектуру ігрової системи з поділом на логічні модулі, відповідальні за управління ресурсами, інвентарем, крафтом, взаємодією та виконанням ігрових завдань. Впровадження власної системи керування залежностями забезпечило слабку зв’язність компонентів, розширюваність архітектури та зручність подальшого розвитку проєкту.

Виконання п’ятого завдання – реалізація оптимізаційних процедур і тестування продуктивності – дозволило підвищити стабільність і швидкодію ігрового продукту. Було застосовано оптимізацію освітлення, системи оклюзії, рівні деталізації моделей, а також проведено профілювання програмних модулів із подальшим рефакторингом критичних ділянок коду. Комплексне тестування підтвердило коректність роботи ігрових механік та ефективність реалізованих оптимізаційних рішень.

Таким чином, у ході виконання кваліфікаційної роботи усі поставлені завдання було виконано в повному обсязі, а отримані результати підтвердили досягнення мети дослідження. Практична цінність роботи полягає у можливості використання розробленої архітектурної моделі та програмних

рішень у навчальному процесі, експериментальних і інді-геймдев проєктах, а також як основи для подальшого розвитку ігрових систем зі складною логікою, розширеним візуальним середовищем та оптимізованою продуктивністю.

У першому розділі здійснено аналіз сучасного стану розвитку ігрової індустрії та інструментів створення цифрових ігор. Охарактеризовано тенденції зростання популярності кросплатформенних рушіїв, застосування відкритих програмних рішень, розширення освітнього потенціалу геймдев-технологій і підвищення ролі інтерактивності та продуктивності у проєктуванні цифрових середовищ.

Проведений аналіз наукових джерел підтвердив конкурентоспроможність Godot Engine серед сучасних інструментів розробки як універсальної платформи, що поєднує відкриту архітектуру, підтримку двовимірних та тривимірних проєктів, гнучку систему сценарного програмування та сумісність з мовою C#. Отриманий теоретичний огляд дозволив обґрунтувати вибір платформи для подальшої практичної реалізації ігрового проєкту.

Впровадження власної системи керування залежностями дозволило централізовано контролювати створення та ініціалізацію компонентів, мінімізувати зв’язність між модулями та забезпечити розширюваність програмної архітектури. Отримані результати підтвердили доцільність застосування принципів абстракції, інкапсуляції та слабкої зв’язності як основи побудови стабільної і масштабованої ігрової платформи.

У другому розділі здійснено проєктування та впровадження ключових ігрових механік. Реалізовано систему виявлення та підбору об’єктів на основі технології RayCast3D, впроваджено механізм візуальної індикації інтерактивних елементів із використанням Outline Shader, створено комплексні системи обробки ресурсів, інвентарю, крафту та управління замовленнями.

Послідовна інтеграція зазначених компонентів дозволила сформувати єдиний технологічний цикл ігрових взаємодій – від збору матеріалів до створення і використання готових виробів. Модульна реалізація рецептів і виробів дозволила передбачити динамічне розширення ігрового контенту без необхідності втручання у базову логіку програми. Візуальні підказки та анімаційні ефекти підвищили наочність процесів і забезпечили інтуїтивність взаємодії користувача з ігровим середовищем.

Профілювання програмних модулів дозволило виявити критичні ділянки обчислень і здійснити їх рефакторинг із застосуванням кешування, оптимальних структур даних та зменшення повторних ресурсозатратних викликів. Реалізація власного DI-контейнера була оптимізована з урахуванням накладних обчислювальних витрат, що дозволило зберегти масштабованість архітектури без втрати швидкодії.

Отримані результати засвідчують практичну цінність виконаної роботи для використання в освітньому процесі підготовки фахівців з інформаційних технологій, а також для застосування у створенні навчальних, експериментальних та інді-геймдев проєктів. Реалізована архітектурна модель та напрацьовані технологічні рішення можуть бути використані як основа для подальшого розвитку ігрових систем зі складною логікою, розширеним візуальним середовищем та оптимізованою продуктивністю.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ангел М. В. Розробка 2D гри на платформі Godot: кваліфікаційна робота на здобуття освітнього ступеня «бакалавр»: спеціальність 121 «Інженерія програмного забезпечення». ЧНУ ім. Петра Могили. Миколаїв. 2024. 79 с.

2. Багатопотоковість. VPN Unlimited. URL: https://www.vpnunlimited.com/ua/help/cybersecurity/multithreading?srsltid=AfmBOoqUGhpNpol7mHq7H1GnqP06i8XbElHdU6pGnnEIrxedtnkZYQtX (дата звернення: 10.10.25).

3. Базюк Р. Л. Розробка відеогри на базі рушія Godot з використанням мови програмування C# : робота на здобуття кваліфікаційного ступеня бакалавра: спец. 122 Комп'ютерні науки; наук. кер. Л. П. Матійчук. Тернопіль: Тернопільський національний технічний університет імені Івана Пулюя. 2025. 57 с.

4. Балицький Н. Технологій ігрових рушіїв. *Інформаційно-комунікаційні технології в освіті*. 2024. № 13. URL: https://e-journals.udu.edu.ua/index.php/ikt/article/view/1453. (дата звернення: 20.02.25).

5. Бойченко М. А. Розробка гри у жанрі Roguelike засобами рушія Godot Engine. Кваліфікаційна робота. Керівник: доц., к. ф.-м. н. О. Ю. Тарасова. Кривий Ріг. 2025. 74 с.

6. Вербовецький Д. В. Використання ігрових засобів під час вивчення курсу «комп'ютерні мережі» у закладах вищої освіти. *Сучасні цифрові технології та інноваційні методики навчання: досвід, тенденції, перспективи*: матеріали XIII Міжнар. наук.-практ. інтернет-конф. (м. Тернопіль, 05 квіт. 2024 р.). Тернопіль: ТНПУ ім. В. Гнатюка, 2024. С. 222–225.

7. Вербовецький Д. М. Методика використання цифрових ігрових технологій у процесі підготовки майбутніх бакалаврів інформатики.

Дисертація. *Національна академія педагогічних наук України*. Київ. 2025. 295 с.

8. Воєвода А. Л. Застосування концепції «Game Based Learning» в освітньому процесі. *Науковий часопис Національного педагогічного університету імені М. П. Драгоманова*. Фізика і математика у вищій і середній школі: зб. наук. праць. Київ : Вид-во НПУ імені М. П. Драгоманова. 2018. Вип. 20. С. 38–44.

9. Джуга Д. Є., Мартинюк С. В. Godot Engine – інструмент для підготовки фахівців у сфері інженерії ігрових проєктів. *Освітні стратегії підготовки фахівців ІТ-галузі*. Сучасні цифрові технології та інноваційні методики навчання: досвід, тенденції, перспективи. 2024. №13. С. 155–157.

10. Oleksiuk V. P., Dzhuha D. Y., Melnyk P. P., Verbovetskyi D. V. Development of the student simulator game: from concept to code. *7th Workshop for Young Scientists in Computer Science & Software Engineering (CS&SW*). Kryvyi Rih: National University, 2024. P. 89–109.

11. Мельник П., Василенко Я. Особливості використання рушія Godot та C# для розробки ігрових застосунків. *Сучасні цифрові технології та інноваційні методики навчання: досвід, тенденції, перспективи*: матеріали XV Міжнародної науково-практичної інтернет-конференції (м. Тернопіль, 10 квітня 2025 р.). Тернопіль: ТНПУ ім. Володимира Гнатюка, 2025. С. 155–159.

12. Мельник П., Василенко Я. Оптимізація продуктивності ігор, створених з використанням рушія godot та C#. *Сучасні цифрові технології та інноваційні методики навчання: досвід, тенденції, перспективи*: матеріали XVI Міжнародної науково-практичної інтернет-конференції (м. Тернопіль, 10 квітня 2025 р.). Тернопіль: ТНПУ ім. Володимира Гнатюка, 2025. С. 246–249.

13. Документація до Godot Engine. Godot Docs – master branch. URL: https://docs.godotengine.org/uk/4.x/index.html (дата звернення: 10.01.25).

14. Документація до Godot Engine. Вступ до Godot. URL: https://docs.godotengine.org/uk/4.x/getting_started/introduction/introduction_to_godot.html (дата звернення: 10.01.25).

15. Документація до Godot Engine. Основи C#. URL: https://docs.godotengine.org/uk/4.x/tutorials/scripting/c_sharp/c_sharp_basics.html (дата звернення: 12.01.25).

16. Загальний огляд мови програмування С. Національний транспортний університет. StudFiles. URL: https://studfile.net/preview/8878391/ (дата звернення: 08.08.25).

17. Історія виникнення програмування. Sutori. URL: https://www.sutori.com/en/story/istoriia-vinikniennia-mov-proghramuvannia--D3K6tckLhc1YY564MFZiKfPN (дата звернення: 02.02.25).

18. Ковальчук М. О. Дослідження можливостей рушія Unreal Engine для створення віртуальних ігрових середовищ: кваліфікаційна робота магістра спеціальності 122 «Комп'ютерні науки». наук. Керівник Н. Р. Полуектова. Запоріжжя: ЗНУ. 2025. 62 с.

19. Легковазна система збереження та завантаження в Godot 4 з C#: практичний посібник. URL: https://javascript.org.ua/legkovazhna-sistema-zberezhennya-ta-zavantazhennya-v-godot-4-z-c-praktichnij-posibnik/ (дата звернення: 24.05.25).

20. Малініч П., Войтко В. Програмна архітектура взаємозв'язку аналітики телеметрії відеоігор та динамічної генерації ігрового контенту. *Herald of Khmelnytskyi National University Technical sciences*. 2025. № 355(4). С. 365–372.

21. Оптимізація ваших проєктів Godot для підвищення ефективності. URL: https://uk.sharpcoderblog.com/blog/optimizing-your-godot-projects-for-performance (дата звернення: 15.04.25).

22. Оптимізація. Таємниця SEO. URL: https://www.taina.com.ua/shho-take-optymizacija (дата звернення: 01.12.25).

23. Основні методи розробки ігор у Godot. Sharp Coder Blog. URL: https://uk.sharpcoderblog.com/blog/essential-techniques-for-game-development-in-godot (дата звернення: 25.03.25).

24. Про мову програмування C#. Foxminded. URL: https://foxminded.ua/prohramuvannia-na-si/ (дата звернення: 28.10.25).

25. Розробка гри на C# WPF. Створюємо гру без ігрового двигуна. itProger. URL: https://itproger.com/ua/news/razrabotka-igri-na-c-wpf-sozdaem-igru-bez-igrovogo-dvizhka (дата звернення: 16.07.25).

26. C#: що це за мова та де її використовують. Robot_dreams. URL: https://robotdreams.cc/uk/blog/284-s-chto-eto-za-yazyk-i-gde-ego-ispolzuyut (дата звернення: 03.09.25).

27. Створення простої мережевої гри в Godot. Fun open source. URL: https://youtu.be/cJNN14W7OEg (дата звернення: 10.05.25).

28. Стрига Д. М. Дослідження використання ігрових рушіїв для ствопення 2D платформеру. Харківський національний університет радіоелектроніки, другий (магістерський) рівень вищої освіти. 2023. 65 с.

29. Фурсова Н. А., Коломієць В. В. Переваги та недоліки використання Godot Engine при розробці кооперативної комп'ютерної гри. *Тези 71-ої наукової конференції професорів, викладачів, наукових працівників, аспірантів та здобувачів університету*. Полтавський національний технічний університет імені Юрія Кондратюка. 2019. Т. 1. С. 373–374.

30. Що таке механіка гри сьогодні? FoxmindEd. URL: https://foxminded.ua/igrova-mekhanika/ (дата звернення: 01.12.25).

31. Alchimowicz S., Plechawska-Wójcik M. Comparative analysis of the implementation performance using selected scripting languages in the Godot game engine. *Journal of Computer Sciences Institute*, 2024. № 31. С. 68–72. DOI: https://doi.org/10.35784/jcsi.5428.

32. Bharambe N. P., Mhaddalkar Y. P., Yeram H. M., Jadhav V. 2D Platformer Game Development with Godot Engine. *International Journal of*

Advance Research, Ideas and Innovations in Technology. IJARIIT. Issue-1, 2025. Vol. 11.

33. Dobroskok I., Rzhevska N., Ayyıldız H., Zaimova D., Zheliazkov G. Game development software tools in higher educational institutions: experience of ukraine, turkey and bulgaria. ITLT, 2020. Vol. 78, № 4. P. 90–104.

34. Flores Jonatan L. Guzman, Cieza-Mostacero Segundo E. Artificial Intelligence in Video Game Development with Accessibilities in Godot Engine. *TEM Journal*. 2025. Vol. 14, № 4. P. 3403–3411.

35. Freeman A., Sanderson S. Pro ASP.NET Core MVC 2. Apress. 2017. 1024 p. URL: https://sandbox.getindico.io/event/1440/attachments/302/440/Freeman_A_-_Pro_ASP_NET_Core_MVC_2_Seventh_Edition_-_2017.pdf (дата звернення: 23.07.25).

36. Freeman E., Robson E. Head first design patterns: building extensible and maintainable object-oriented software. 2020. 694 p.

37. Google trends. URL: https://trends.google.com.ua/trends/explore?date=today%205-y&q=Godot&hl=uk (дата звернення: 23.07.25).

38. Gregory J. Game Engine Architecture. CRC Press. 2018. 1240 p.

39. Hejlsberg A. The C# Programming Language. *Addison-Wesley*. 2006. 704 p.

40. Holfeld J. On the relevance of the Godot Engine in the indie game development industry. University of Kassel. 2023. P. 1–9. URL: https://www.researchgate.net/publication/383116776_On_the_relevance_of_the_Godot_Engine_in_the_indie_game_development_industry (дата звернення: 20.02.25).

41. Learn 2D Game Development: Godot 4.3 + C# from Scratch. URL: https://www.udemy.com/course/learn-2d-game-development-godot-43-c-from-scratch/ (дата звернення: 20.02.25).

42. Linietsky J. A decade in retrospective and future. 2019. URL: https://godotengine.org/article/retrospective-and-future.

43. Masood Danial M. «Comparison of Programming Languages in Game Development». 2020. P. 19–1210.

44. Mono Project. Mono Documentation. URL: https://www.mono-project.com (дата звернення: 14.06.25).

45. Most used programming languages among developers worldwide as of 2024. Statista. URL: https://www.statista.com/statistics/793628/worldwide-developer-survey-most-used-languages/ (дата звернення: 04.11.25).

46. Ranaweera M., Mahmoud Q. H. Deep reinforcement learning with Godot Game Engine, 2024. № 13(5). P. 985. DOI: https://doi.org/10.3390/electronics13050985.

47. Salmela T. Game development using the open-source Godot Game Engine. *Degree Programme in Media and Arts Interactive Media*. 2022. 67 p. URL: https://www.theseus.fi/bitstream/handle/10024/746943/Salmela_Tero.pdf?sequence=3&isAllowed=y

48. Sultan A. A. Generative AI in Game Design: Enhancing Creativity or Constraining Innovation? Journals J. Intell. College of Computer Science and Engineering, University of Jeddah. Saudi Arabia. 2025. Vol. 13, № 6(60).

49. The Best Gaming Engines for 2024 – Incredibuild. URL: https://www.incredibuild.com/blog/top-gaming-engines-you-should-consider (дата звернення: 10.09.25).

50. The Godot Engine: A New Era in Open-Source Gaming. Medium. URL: https://medium.com/%40foobar404/the-godot-engine-a-new-era-in-open-source-gaming-047d4b4c784f (дата звернення: 17.04.25).

51. Thorn A. Scripting with C# in Godot: Common Tasks. In: Moving from Unity to Godot. Apress, Berkeley, CA. 2020. URL: https://link.springer.com/chapter/10.1007/978-1-4842-5908-5_3 (дата звернення: 13.06.25).

52. Unity Real-Time Development Platform. 3D, 2D, VR & AR Engine. URL: https://unity.com (дата звернення: 13.09.25).

53. Unreal Engine: The most powerful real-time 3D creation tool. URL: https://www.unrealengine.com (дата звернення: 18.10.25).

54. Vitvytska S. S., Smailova T. U. Application of game technologies in the training of higher education students for pedagogical activities Zhytomyr Ivan Franko State University Journal. Pedagogical Sciences. 2025. Vol. 1(120). P. 292–301.

55. Vuorela J. Differences Between C# and GDScript in Godot Game Engine. Business information systems games production. *Bachelor's thesis*. Tampere University of Applied Science. 2024. 35 p.

56. Winata L., Maulana M. A., Susilo J. Studi Perbandingan Pengembangan Game dalam GDScript dengan Godot dan C# dengan Unity. 2025. Vol. 7, № 3. P. 715–721.

57. Your free, open-source game engine. URL: https://godotengine.org/ (дата звернення: 11.08.25).