Plag

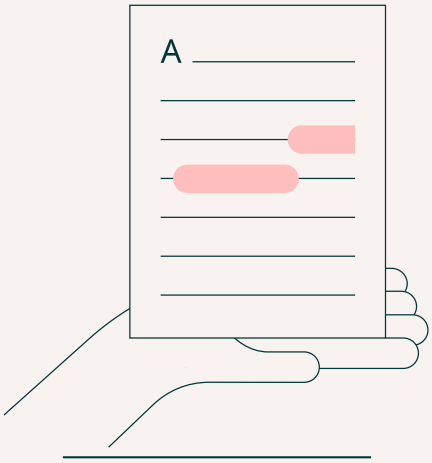
Звіт про оригінальність

● Оцінка схожості 2%

● Ризик плагіату СЕРЕДНІЙ

Ігор Кагало 2025-06-06 21:58

Посилання на звіт: ZUHa / Посилання користувача: qfC8



Ось вона –
Ваша звіт про оригінальність!

Ми раді повідомити, що перевірка вашого документа завершена, і результати вже готові! Наші алгоритми старанно працювали, щоб знайти збіги в наших базах даних.

На наступних сторінках ви знайдете результати перевірки:

Бали
Збіги
Посилання

Ваш документ було перевірено за такими джерелами:

- ✓ База даних інтернет-джерел
- ✓ База даних наукових статей
- ○ Глибока перевірка (наш вдосконалений алгоритм)

Бали

Збіги тексту	2%
Перефразування	0%
Цитований текст	0%
Неправильне цитування	0%
Збігів не знайдено	98%

Ризик плагіату **СЕРЕДНІЙ**

Ризик плагіату вказує, як збіги тексту розподілені по документу. Вищий ризик виникає, коли збіги з'являються близько один до одного, наприклад, у тому самому абзаці або розділі.

Оцінка схожості 2%

Оцінка схожості показує, скільки слів або символів у вашому документі збігаються з текстами інших документів, включаючи перефразовані тексти або неправильні цитати.

Збіги

ВСТУП

Сучасна індустрія комп'ютерних ігор постійно розвивається, впроваджуючи нові технології та підвищуючи рівень реалізму та інтерактивності. Одним із ключових аспектів створення якісних ігрових продуктів є використання потужних ігрових рушіїв, таких як Unity, які надають розробникам широкий спектр інструментів для реалізації складних механік та мультиплеєрних режимів.

Метою даного дослідження є розробка гри «TANKI 3D» – аркадного танкового шутера для двох гравців з локальним мультиплеєром. Основним завданням було створення механік руху танків, системи пострілів, фізики рикошетів, інтерактивного інтерфейсу та локалізації.

Об'єктом дослідження виступив процес розробки 3D–гри з використанням рушія Unity, включаючи моделювання фізики та інтеграцію користувацького інтерфейсу. Для досягнення мети використовувались аналіз існуючих ігрових механік, експериментування з фізичними параметрами в Unity та ітеративний процес розробки.

Робота 14 складається з трьох розділів: теоретичні основи комп'ютерних ігор та ігрових рушіїв, технології розробки Unity та його архітектура, практична реалізація гри «TANKI 3D» з усіма механіками та інтерфейсом. Фактологічною основою дослідження стали офіційна документація Unity, наукові публікації з розробки ігор та власний досвід роботи з ігровими рушіями.

1 ТЕОРЕТИЧНІ ОСНОВИ КОМП'ЮТЕРНИХ ІГОР

1.1 Еволюція ігрових рушіїв

Ігрові рушії, як основа для створення комп'ютерних ігор, пройшли значний еволюційний шлях від простих програмних бібліотек до складних інтегрованих середовищ з реалістичною графікою, фізикою та інструментами для розробки. Історично розвиток ігрових рушіїв 11 можна умовно поділити на кілька ключових етапів.

Зародження ігрових рушіїв відбулося в академічному середовищі 1970–1980–х років. Одним із перших прикладів можна вважати набір бібліотек, використаних для створення гри «Spacewar!» у 1962 році на міні–комп'ютері PDP–1. Хоча це не був повноцінний рушій у сучасному розумінні, він заклав основи для модульного підходу до розробки ігор. На рисунку 1.1 представлена гра «Spacewar!» на комп'ютері PDP–1, 1962 рік.

Рисунок 1.1 – Гра «Spacewar!» на комп'ютері PDP–1, 1962 рік

У 1970–х роках з'явилися перші комерційні ігрові автомати, такі як «Pong» (1972) від Atari. Ці ігри використовували власні пропрієтарні рушії, оптимізовані для конкретного апаратного забезпечення. На рисунку 1.2 представлений аркадний автомат з грою «Pong».

Рисунок 1.2 – Аркадний автомат з грою «Pong»

У 1972 році компанія Atari випустила одну з перших аркадних відеоігор – Pong, яка моделювала настільний теніс. Цей автомат став проривом у світі цифрових розваг і започаткував масове виробництво аркадних ігор. Гравці керували платформами за допомогою поворотних ручок, відбиваючи м’яч на екрані, що стало простим, але захопливим ігровим процесом для тогочасної публіки.

1980–ті роки ознаменувалися появою перших персональних комп'ютерів, таких як Apple II та Commodore 64. Розробники почали створювати ігри з використанням власних рушіїв, написаних на мовах програмування, таких як BASIC та Assembler. На рисунку 1.3 представлені популярні аркадні ігри початку 1980–х років.

Рисунок 1.3 – Популярні аркадні ігри початку 1980–х років

1990–ті роки стали періодом розвитку 3D–графіки. Ігри, такі як «Doom» (1993) та «Quake» (1996), використовували власні 3D–рушії, які заклали основи для сучасних технологій рендерингу. На рисунку 1.4 представлені ігри, які започаткували епоху 3D–графіки та задали стандарти жанру FPS.

Рисунок 1.4 – Ігри які започаткували епоху 3D–графіки та задали стандарти жанру FPS

2000–ті роки ознаменувалися появою перших комерційних ігрових рушіїв, таких як Unreal Engine та Unity. Ці рушії пропонували розробникам широкий набір інструментів для створення ігор, включаючи графічний рендеринг, фізику, штучний інтелект та редактор рівнів.

2010–ті роки принесли розвиток мобільних ігор та віртуальної реальності. Ігрові рушії, такі як Unity, стали доступними для широкого кола розробників, дозволяючи

створювати ігри для різних платформ.

2020–ті роки характеризуються розвитком технологій трасування променів, 3 штучного інтелекту та хмарного геймінгу. Ігрові рушії, такі як Unreal Engine 5, пропонують розробникам інструменти для створення фотореалістичних ігор з відкритим світом.

Сьогодні ігрові рушії є основою для створення більшості комп'ютерних ігор. Вони постійно розвиваються, пропонуючи розробникам нові інструменти та технології для створення все більш реалістичних та захоплюючих ігор.

1.2 Класифікація ігрових рушіїв

Класифікація ігрових рушіїв є важливим інструментом систематизації, що дозволяє структурувати різноманітне програмне забезпечення 3 для розробки ігор за спільними функціями, можливостями та особливостями використання. Упродовж розвитку ігрової індустрії сформувалася розгалужена система класифікації рушіїв, яка продовжує еволюціонувати з появою нових технологій та підходів до розробки ігор.

2D–рушії – рушії, орієнтовані на розробку ігор з двовимірною графікою. Вони часто пропонують інструменти для роботи зі спрайтами, тайловими картами, 2D–фізикою та іншими елементами 2D–ігор.

Приклад: 3 Godot Engine – безкоштовний 3 та відкритий 3 рушій, який підтримує як 2D, так і 3D розробку, але має потужні інструменти саме для 2D–ігор. На рисунку 1.5 представлений інтерфейс Godot Engine.

2 Рисунок 1.5 – Інтерфейс Godot Engine

2 3D–рушії – рушії, призначені для створення ігор з тривимірною графікою. Вони забезпечують розробників інструментами 2 для роботи з 3D–моделями, текстурами, освітленням, 3D–фізикою та іншими аспектами 3D–розробки.

Приклад: 3 Unreal Engine – один з найпотужніших та найпопулярніших 3D–рушіїв, який 2 використовується для створення AAA–ігор, фільмів та інших проєктів з високими вимогами до графіки та продуктивності. На рисунку 1.6 представлений 2 інтерфейс Unreal Engine.

2 Рисунок 1.6 – Інтерфейс Unreal Engine

2 Універсальні рушії – рушії, які підтримують 3 розробку як 2D, так і 3D ігор, надаючи розробникам гнучкість у виборі стилю та технології для своїх проектів. Приклад: Unity – надзвичайно популярний рушій, який дозволяє створювати ігри для широкого спектру

платформ, від мобільних пристроїв до ПК та консолей, як у 2D, так і в 3D. На рисунку 1.7 представлений інтерфейс Unity.

Рисунок 1.7 2 – Інтерфейс Unity

2 Спеціалізовані рушії – рушії, розроблені для конкретних жанрів або типів ігор, пропонуючи оптимізовані інструменти та функції для цих завдань. Приклад: Ren'Py – рушій, спеціально створений для розробки візуальних новел, пропонуючи спрощені інструменти 2 для роботи з текстом, зображеннями та інтерактивними елементами. На рисунку 1.8 представлений інтерфейс Ren'Py.

Рисунок 1.8 – Інтерфейс Ren'Py

Відкриті рушії – рушії з відкритим вихідним кодом, які дозволяють розробникам модифікувати та розширювати їх функціональність відповідно до своїх потреб. Приклад: Godot Engine (згаданий вище) є також прикладом відкритого рушія, 2 що робить його гнучким та адаптованим до різних проектів. Важливо зазначити, що класифікація ігрових рушіїв не є статичною, і з розвитком технологій та появою нових підходів до розробки ігор межі між різними категоріями можуть розмиватися.

1.3 Особливості розвитку індустрії ігрових рушіїв

Індустрія ігрових рушіїв демонструє динамічний розвиток та стрімке зростання, що супроводжується унікальними бізнес–моделями, структурними трансформаціями та соціокультурними впливами. Аналіз основних тенденцій та особливостей розвитку цієї галузі дозволяє краще зрозуміти її поточний стан та прогнозувати майбутні напрямки еволюції.

Ринок ігрових рушіїв є важливою складовою частиною ширшої індустрії відеоігор. Хоча точні цифри щодо обсягу ринку саме ігрових рушіїв можуть бути складними для визначення через їх інтегрованість у вартість розробки ігор, можна відзначити кілька ключових тенденцій. Зі зростанням загального ринку відеоігор зростає і попит на ігрові рушії. У 2023 році глобальний ринок відеоігор досяг обсягу в 184,4 мільярда доларів, що створює сприятливе середовище для розвитку індустрії ігрових рушіїв.

Ринок ігрових рушіїв значною мірою контролюється кількома великими гравцями, такими як Epic Games (Unreal Engine) та Unity Technologies (Unity). Їхні рушії використовуються для створення широкого спектру ігор, від ААА–проєктів до мобільних ігор. Компанії, що розробляють ігрові рушії, використовують різні моделі монетизації, включаючи ліцензування, підписки, роялті від продажів ігор, а також надання додаткових послуг, таких як хмарні сервіси та інструменти для розробників.

Індустрія ігрових рушіїв також зазнає трансформації бізнес–моделей. Подібно до моделі

«Програмне забезпечення як послуга» (SaaS), ігрові рушії все частіше пропонуються за підпискою, що дозволяє розробникам отримувати доступ до найновіших функцій та оновлень, а також спрощує процес розробки. Компанії активно розширюють свої екосистеми, пропонуючи розробникам додаткові інструменти, плагіни, асети та сервіси, створюючи більш комплексну та зручну платформу для розробки ігор.

Значного розвитку набули магазини асетів, такі як Unreal Engine Marketplace та Unity Asset Store. Вони дозволяють розробникам купувати та продавати готові моделі, текстури, скрипти та інші ресурси, що значно прискорює процес розробки. Структура індустрії ігрових рушіїв також зазнає змін. Спостерігається консолідація ринку, коли великі компанії, такі як Epic Games та Unity Technologies, поглинають менші студії та розробників технологій, розширюючи свої можливості та доступ до нових технологій.

Значення вертикальної інтеграції також зростає. Компанії, що розробляють ігрові рушії, все частіше інтегрують свої технології з іншими аспектами розробки ігор, такими як хмарні сервіси, інструменти для розробки мультиплеєра та платформи дистрибуції. Поряд з великими комерційними рушіями розвивається сегмент інді–рушіїв, які пропонують розробникам доступні та гнучкі інструменти для створення ігор з обмеженим бюджетом.

Індустрія ігрових рушіїв демонструє тенденції до глобалізації та локалізації. Такі рушії, як Unity та Unreal Engine, доступні розробникам по всьому світу, що сприяє глобалізації індустрії відеоігор. Водночас компанії все більше уваги приділяють локалізації своїх інструментів та документації для різних регіонів та мов, щоб зробити їх більш доступними для розробників у різних країнах. Розробникам також пропонуються інструменти та функції для підтримки регіональних особливостей, таких як мовні версії, культурні аспекти та технічні вимоги різних платформ.

Ігрові рушії мають значний соціокультурний вплив. Вони надають розробникам інструменти для створення інтерактивних розваг, освітніх програм, симуляцій та інших видів контенту, що сприяє розвитку творчості та інновацій. Активні спільноти розробників, що обмінюються знаннями, досвідом та ресурсами, сприяють розвитку професійних навичок та співпраці. Крім того, технології, розроблені для ігрових рушіїв, знаходять застосування в інших сферах, таких як кіно, архітектура, інженерія та медицина, демонструючи їх широкий потенціал.

Разом з тим, розвиток індустрії супроводжується формуванням складної технологічної екосистеми. З’являються компанії, що пропонують спеціалізовані технології та сервіси для ігрових рушіїв, такі як інструменти для оптимізації продуктивності, плагіни для рендерингу, системи для розробки мультиплеєра та інструменти для штучного інтелекту. Ігрові рушії все частіше використовуються для створення контенту для різних

медіа, таких як фільми, телебачення та віртуальна реальність, що сприяє кросмедійній інтеграції та розширює можливості використання рушіїв.

Попри успішний розвиток, індустрія ігрових рушіїв стикається з низкою викликів. Розробники ігрових рушіїв постійно працюють над подоланням технологічних обмежень, таких як продуктивність, масштабованість та сумісність з різними платформами. Ринок ігрових рушіїв є висококонкурентним, і 6 компанії повинні постійно інвестувати в дослідження та розробки, щоб залишатися конкурентоспроможними. Також важливим фактором є якісна підтримка розробників, надання їм навчальних матеріалів та ресурсів для ефективного використання рушіїв.

Таким чином, розвиток індустрії ігрових рушіїв характеризується комплексними трансформаціями на різних рівнях – від бізнес–моделей та ринкової структури до соціокультурного впливу та технологічних інновацій. Розуміння цих процесів є важливим не лише для учасників індустрії, але й для дослідників цифрової економіки та сучасної культури загалом.

1.4 Технологічні тренди в ігрових рушіях

Сучасні ігрові рушії перебувають на перетині мистецтва та високих технологій, постійно впроваджуючи інноваційні рішення для підвищення якості ігрового досвіду, оптимізації процесів розробки та розширення можливостей взаємодії з гравцями. Аналіз технологічних трендів 15 дозволяє не лише відстежувати поточний стан галузі, але й прогнозувати напрямки її подальшого розвитку.

Однією з найбільш динамічних областей розвитку є технології рендерингу, що дозволяють створювати все більш реалістичні візуальні ефекти. Трасування променів (Ray Tracing) стало ключовою технологією, що дозволяє моделювати проходження світла в реальному часі, створюючи фотореалістичні відображення, тіні та заломлення світла.

Unreal Engine 5 використовує трасування променів для створення надзвичайно реалістичного освітлення та відображень. На рисунку 1.9 представлено приклад трасування променів в Unreal Engine 5.

Рисунок 1.9 – Трасування променів в Unreal Engine 5

Паралельно розвивається технологія 4 DLSS (Deep Learning Super Sampling) від NVIDIA та FSR (FidelityFX Super Resolution) від AMD, які використовують штучний інтелект для масштабування зображення з нижчої роздільної здатності до вищої, дозволяючи досягнути значного підвищення продуктивності без помітної втрати якості зображення.

Ці технології інтегруються в ігрові рушії 4 для покращення продуктивності рендерингу.

Global Illumination (глобальне освітлення) та об'ємні ефекти (volumetric effects) також стали стандартом для AAA–проектів, дозволяючи відтворювати реалістичні атмосферні явища, такі як туман, дим, розсіяне світло в повітрі тощо.

Unity використовує глобальне освітлення для створення реалістичних сцен з динамічним освітленням. На рисунку 1.10 представлено приклад глобального освітлення в Unity.

Рисунок 1.10 – Глобальне освітлення в Unity

Штучний інтелект трансформує ігрові рушії, дозволяючи створювати процедурно згенеровані рівні, текстури та моделі. Завдяки генеративним нейронним мережам ігрові світи стають унікальними, а персонажі отримують складніші сюжетні лінії. Нелінійна поведінка неігрових персонажів забезпечує більш реалістичну взаємодію, використовуючи дерева рішень і машинне навчання. Динамічне балансування складності підлаштовує гру під стиль гравця, створюючи оптимальний досвід. Інтеграція технологій мовлення та розпізнавання обличчя покращує діалогові системи та анімацію.

Хмарний геймінг спрощує доступ до ігор, обробляючи рендеринг у хмарі. Платформи як Google Stadia та NVIDIA GeForce Now дозволяють грати на слабких пристроях без втрати продуктивності. Ігрові рушії адаптуються для потокової трансляції контенту. Unity та Unreal Engine вдосконалюються, спрощуючи розробку для інді–розробників і великих студій. Крос–платформна розробка дозволяє запускати ігри на ПК, консолях, мобільних пристроях і VR–системах з мінімальними змінами коду.

Технології VR та AR розширюють можливості занурення в ігровий процес. Сучасні гарнітури, такі як Oculus Quest 2 та Valve Index, забезпечують високу деталізацію, широке поле зору та точне відстеження рухів. Ігрові рушії пропонують інструменти для створення VR/AR–додатків. Фізичні симуляції роблять ігри більш реалістичними: моделюється руйнування об’єктів, реалістична гідродинаміка, анімація тканин і волосся, процедурна анімація рухів.

Нові технології, такі як блокчейн та NFT, впроваджують концепцію цифрової власності. Завдяки 5G мобільні ігри стають швидшими, а багатокористувацькі проєкти – стабільнішими. Біометричні інтерфейси аналізують фізіологічний стан гравця для адаптації геймплею. Системи захоплення руху вдосконалюються, дозволяючи передавати акторську гру з найдрібнішими деталями. WebGL та HTML5 забезпечують створення браузерних ігор без необхідності встановлення додаткового ПЗ.

Ігрова індустрія є рушієм технологічного прогресу. Розробники знаходять нові алгоритми оптимізації, що згодом застосовуються в науці, архітектурі та промисловому

дизайні. Завдяки цьому ігрові рушії стають дедалі потужнішими, відкриваючи нові горизонти у сфері цифрових технологій.

2 ТЕХНОЛОГІЇ ТА ІНСТРУМЕНТИ РОЗРОБКИ

2.1 Середовище розробки Unity

Unity – це крос–платформне середовище розробки комп'ютерних ігор, створене компанією Unity Technologies. З моменту свого заснування у 2005 році Unity швидко здобула популярність серед розробників різного рівня – від інді–команд до великих студій. Станом на 2025 рік, Unity використовується для створення понад 50% усіх мобільних ігор та має значну частку на ринку ПК та консольних розробок.

Однією з головних переваг Unity є інтуїтивний інтерфейс користувача. Інтегроване середовище розробки (IDE) Unity має візуальний редактор сцен, що дозволяє розміщувати та налаштовувати об'єкти у двовимірному або тривимірному просторі без необхідності написання коду. Це значно спрощує процес розробки, особливо для початківців.

Архітектура Unity базується на принципі компонентно–орієнтованого програмування. Кожен ігровий об'єкт (GameObject) може мати довільний набір компонентів, які визначають його поведінку. Такий підхід забезпечує високу гнучкість та дозволяє розробникам створювати складні системи без необхідності переписувати код для кожного елемента гри.

Unity підтримує крос–платформну розробку, дозволяючи створювати ігри для понад 25 платформ, серед яких Windows, macOS, Linux, iOS, Android, PlayStation, Xbox, Nintendo Switch, WebGL та VR/AR–пристрої. Зміна цільової платформи зазвичай не вимагає значних модифікацій вихідного коду, що робить Unity зручним вибором для розробників, які прагнуть охопити широку аудиторію.

Важливу роль у розвитку Unity відіграє офіційний магазин ресурсів Asset Store. У ньому розробники можуть придбати та продати готові компоненти, моделі, текстури, звуки, шейдери та інші ресурси, що значно прискорює процес створення ігор.

Документація Unity є однією з найбільш розгорнутих серед середовищ розробки ігор. Завдяки великій базі навчальних матеріалів та активній спільноті розробників новачки можуть швидко освоїтися в системі, а досвідчені фахівці – знайти рішення для складних завдань.

Сучасні версії Unity включають вдосконалену систему рендерингу Universal Render Pipeline (URP) та High Definition Render Pipeline (HDRP), інструменти для створення мультиплеєрних ігор (Unity Netcode for GameObjects), покращену підтримку 2D–графіки

та фізики, а також інтеграцію з хмарними сервісами. Постійний розвиток цієї платформи робить її одним із провідних інструментів у світі ігрової індустрії.

2.2 Архітектура та можливості ігрового рушія

Архітектура Unity побудована на модульному принципі, де різні підсистеми рушія взаємодіють між собою для забезпечення повного циклу розробки та функціонування гри.

Ядро рушія (Core Engine) включає низькорівневі компоненти, що забезпечують базову функціональність. До них належить система управління пам'яттю, багатопотокова обробка, система подій та управління ресурсами.

Рушій рендерингу (Rendering Engine) 5 відповідає за відображення графічних елементів на екрані. Він підтримує різні графічні API, такі як DirectX, OpenGL, Vulkan та Metal. Конвеєр рендерингу забезпечує налаштовані етапи обробки зображення, систему шейдерів та матеріалів, а також функції освітлення, тіней та пост–обробки.

Фізичний рушій (Physics Engine) реалізує симуляцію фізичних взаємодій. Він включає систему твердих тіл (Rigidbody), механізм виявлення зіткнень, симуляцію тканин та м'яких тіл, а також шарнірні з'єднання та пружини.

Аудіо рушій (Audio Engine) відповідає за відтворення та обробку звуку. Він підтримує просторове аудіо, міксування, ефекти, а також компресію та декомпресію аудіофайлів.

Система скриптів (Scripting System) дозволяє програмувати логіку гри. Unity використовує 1 мову програмування C# та середовище виконання .NET. Для розширення функціональності можливе використання нативного коду через плагіни.

5 Система анімації (Animation System) забезпечує реалістичну анімацію ігрових об'єктів. Вона підтримує кісткову анімацію, морфінг (Blend Shapes), систему Timeline для синхронізації анімації з подіями, а також процедурну анімацію.

Система UI (User Interface System) включає інструменти для створення користувацького інтерфейсу. Вона базується на Canvas, підтримує адаптивний дизайн та систему подій для взаємодії з користувачем.

Система навігації (Navigation System) забезпечує функціонал для переміщення об'єктів. Використання NavMesh дозволяє реалізовувати пошук шляху, уникнення перешкод та формування груп об'єктів.

Мережевий модуль (Networking Module) забезпечує багатокористувацьку взаємодію. Він містить високорівневі API для синхронізації стану, управління підключеннями та

узгодження 5 даних між клієнтами.

5 Unity використовує архітектурний патерн «Entity–Component–System» (ECS), що забезпечує модульність і високу продуктивність. У цій моделі сутності (Entity) є контейнерами, компоненти (Component) додають функціональність, а системи (System) керують їхньою поведінкою. Такий підхід дозволяє легко 13 комбінувати різні елементи для створення унікальної ігрової 13 логіки.

1 2.3 1 Система компонентів Unity

1 Система компонентів є фундаментальною концепцією архітектури Unity, що реалізує принципи компонентно–орієнтованого програмування. У цій парадигмі функціональність ігрових об'єктів формується шляхом комбінування різних компонентів, замість створення складних ієрархій класів.

Основні елементи системи компонентів Unity:

GameObject – це базовий контейнер у сцені, який сам по собі не має визначеної поведінки або властивостей. Кожен GameObject має лише трансформацію (Transform), що визначає його позицію, обертання та масштаб у просторі;

Component – це функціональний модуль, що додається до GameObject для надання йому конкретних можливостей. Компоненти реалізуються як класи, що успадковуються від базового класу MonoBehaviour;

MonoBehaviour – базовий клас для скриптів, що взаємодіють із системою подій Unity.

Він надає доступ до основних методів життєвого циклу:

Awake() – викликається при ініціалізації скрипта, до початку першого кадру;

Start() – викликається перед першим кадром оновлення;

Update() – викликається кожен кадр;

FixedUpdate() – викликається через фіксовані інтервали часу, використовується для фізичних розрахунків;

LateUpdate() – викликається після всіх оновлень, використовується для завершальних операцій кадру.

Переваги компонентної системи Unity:

Модульність – кожен компонент відповідає за окрему функціональність, що полегшує розробку та тестування;

Повторне використання – компоненти можуть бути використані для різних ігрових об'єктів без дублювання коду;

Гнучкість – функціональність ігрового об'єкта може бути змінена додаванням або видаленням компонентів під час виконання;

Інкапсуляція – кожен компонент інкапсулює свою логіку та дані, що сприяє чистоті коду.

Стандартні компоненти Unity включають:

Renderer (MeshRenderer, SpriteRenderer) – для відображення графіки;

Collider (BoxCollider, SphereCollider) – для визначення зіткнень;

Rigidbody – для симуляції фізики;

Animator – для управління анімаціями;

Audio Source – для відтворення звуків;

Camera – для визначення точки зору гравця;

Light – для освітлення сцени.

Взаємодія між компонентами може здійснюватися кількома способами:

Пряме посилання – один компонент отримує посилання на інший через інспектор або методом GetComponent<T>();

Системи подій – компоненти можуть обмінюватися повідомленнями через події C# або Unity Event System;

Патерн «Спостерігач» – компоненти можуть підписуватися на події інших компонентів;

Сервісні локатори – централізований доступ до спільних сервісів;

1 Система компонентів Unity є потужним інструментом для створення складних ігрових систем з модульної, легко керованої кодової бази.

2.4 Графічні та фізичні можливості

Архітектура Unity побудована на модульному принципі, де різні підсистеми рушія взаємодіють між собою для забезпечення повного циклу розробки та функціонування гри. Основні компоненти архітектури Unity включають ядро рушія, рушій рендерингу, фізичний рушій, аудіо рушій, систему скриптів, систему анімації, систему UI, систему навігації та мережевий модуль.

Ядро рушія відповідає за низькорівневі компоненти, що забезпечують базову функціональність, зокрема систему управління пам'яттю, багатопотокову обробку, систему подій та управління ресурсами. Рушій рендерингу забезпечує відображення графічних елементів на екрані, підтримуючи різні графічні API, конвеєр рендерингу, систему шейдерів, освітлення та тіні, а також пост–обробку зображення.

Фізичний рушій дозволяє симулювати фізичні взаємодії, зокрема систему твердих тіл, систему виявлення зіткнень, симуляцію тканин та м'яких тіл, а також шарнірні з'єднання та пружини. Аудіо рушій забезпечує відтворення та обробку звуку, включаючи просторове аудіо, міксування, ефекти, а також компресію та декомпресію аудіо.

Система скриптів дозволяє програмувати логіку гри, підтримує мову C#, середовище виконання .NET і взаємодію з нативним кодом через плагіни. Система анімації відповідає за анімацію ігрових об'єктів, підтримує кісткову анімацію, морфінг, систему Timeline для синхронізації анімації з іншими подіями, а також процедурну анімацію.

Система UI забезпечує створення користувацького інтерфейсу на основі Canvas, адаптивний дизайн, систему подій та взаємодії. Система навігації дозволяє переміщувати об'єкти за допомогою NavMesh, уникати перешкод та формувати групи об'єктів. Мережевий модуль відповідає за багатокористувацьку взаємодію, синхронізацію стану, управління підключеннями та узгодження даних між клієнтами.

Unity використовує архітектурний патерн «Entity–Component–System» (ECS), де сутність (Entity) є базовим контейнером (GameObject), компонент (Component) представляє функціональні модулі, що додаються до сутностей, а система (System) містить логіку, що обробляє групи компонентів. Така архітектура забезпечує гнучкість та модульність розробки, дозволяючи легко комбінувати різні функціональні блоки для створення унікальної поведінки ігрових об'єктів.

Unity надає розробникам широкий спектр інструментів для створення візуально привабливих проектів та реалістичних фізичних взаємодій. Графічні можливості рушія включають різні системи рендерингу, освітлення, тіней, ефектів пост–обробки, шейдерів і системи частинок.

Рендеринг в Unity може виконуватися за допомогою Built–in Render Pipeline, Universal Render Pipeline (URP) або High Definition Render Pipeline (HDRP). Освітлення реалізується через Global Illumination, Realtime Lighting, Baked Lighting та Mixed Lighting. Система тіней підтримує Shadow Maps, Contact Shadows та Screen Space Shadows. Для створення реалістичних ефектів використовуються Bloom, 12 Depth of Field, Screen Space Reflections, Ambient Occlusion, Motion Blur, HDR та Tonemapping. Шейдерна система включає Shader Graph, Surface Shaders, Vertex/Fragment Shaders і Compute Shaders.

1 Для роботи з частинками застосовуються Visual Effect Graph, Particle System і GPU Instancing.

Фізичні можливості Unity охоплюють систему твердих тіл, систему зіткнень, систему тканин і м'яких тіл, систему шарнірів та з'єднань, а також спеціалізовані фізичні системи. Тверді тіла використовують компонент Rigidbody, підтримують Constraints і Continuous Detection для точного виявлення зіткнень. Для обробки зіткнень застосовуються примітивні колайдери (Box, Sphere, Capsule), Mesh Collider, Compound Collider і Trigger Zones.

Система тканин і м'яких тіл забезпечує симуляцію тканин за допомогою Cloth Component та деформацію моделей через Skinned Mesh Renderer. 1 Для роботи зі з'єднаннями передбачені Hinges, Springs, Fixed Joints та Character Joints. Спеціальні фізичні системи включають Wheel Collider для транспорту, Character Controller для персонажів, Terrain Collider для ландшафту та Articulation Body для точної симуляції роботизованих механізмів.

Фізичні матеріали (Physic Material) дозволяють налаштовувати тертя та пружність поверхонь, а система шарів зіткнень контролює взаємодію між різними об'єктами. Unity постійно розширює свої графічні та фізичні можливості, впроваджуючи нові технології та оптимізуючи існуючі, що дозволяє створювати високоякісні проекти з реалістичною фізикою та графікою.

2.5 Система контролю версій Git

Git є однією з найбільш популярних розподілених систем контролю версій, яка широко використовується в індустрії програмного забезпечення, зокрема в розробці ігор. Вона дозволяє зручно відстежувати зміни в проекті, спрощує командну роботу та надає механізм захисту від втрати даних. Основні поняття Git:

Репозиторій (Repository) – сховище для файлів проекту, яке зберігає всі зміни та історію розробки. Репозиторії поділяються на локальні, які зберігаються на комп'ютері розробника, та віддалені, що зберігаються на сервері (наприклад, GitHub, GitLab);

Комміт (Commit) – фіксація стану проекту на конкретний момент часу. Кожен комміт має унікальний ідентифікатор та включає зміни у файлах, повідомлення про зміни, а також інформацію про автора та час;

Гілка (Branch) – незалежна лінія розробки в рамках проекту, що дозволяє паралельно працювати над новими функціями або експериментувати без ризику для основного коду;

Злиття (Merge) – процес об'єднання змін з різних гілок проекту;

Конфлікт (Conflict) – ситуація, коли зміни в різних гілках несумісні між собою і потребують ручного вирішення.

Налаштування Git для Unity проектів. Для інтеграції Git з Unity необхідно правильно налаштувати інструмент. Ось кілька основних аспектів:

Файл .gitignore – це конфігураційний файл, що визначає, які файли проекту не повинні відслідковуватись системою контролю версій. Для Unity зазвичай ігноруються тимчасові файли, бібліотеки, скомпільовані файли та інші дані, що не є частиною вихідного коду.

Типовий вміст файлу .gitignore для Unity.

Unity–специфічні ігноровані файли

[Aa]ssets/AssetStoreTools

[Bb]uild/

[Ll]ibrary/

[Ll]ocal[Cc]ache/

[Oo]bj/

[Tt]emp/

[Uu]nityGenerated/

Файли про збій

sysinfo.txt

Meta файли Unity3D

.pidb.meta

Файли проектів Visual Studio та інших IDE

[Ee]xportedObj/

.booproj

.csproj

.sln

.suo

Налаштування Unity 1 для роботи з Git:

В налаштуваннях Unity необхідно увімкнути «Visible Meta Files» у розділі «Version Control Mode»;

Рекомендується використовувати «Force Text» для режиму серіалізації активів, щоб файли були зручними для відстеження змін.

Git LFS (Large File Storage) – для ефективної роботи з великими файлами, такими як текстури, моделі або аудіофайли, використовується Git LFS. Це 16 дозволяє зберігати великі бінарні файли поза основним репозиторієм, що покращує продуктивність та зменшує розмір основного репозиторію. Для налаштування Git LFS, потрібно виконати команди:

git lfs install;

git lfs track «.psd»;

git lfs track «.tga»;

git lfs track «.tif»;

git lfs track «.png»;

git lfs track «.fbx»;

git lfs track «.wav»;

git lfs track «.mp3».

У процесі розробки з Git основні команди включають:

Ініціалізація репозиторію – для створення нового репозиторію використовують команду git init, а для клонування існуючого репозиторію – git clone <url>;

Робота зі змінами – щоб перевірити статус файлів використовують команду git status, для додавання змін до репозиторію – git add ., а для фіксації змін з повідомленням – git commit –m «Опис змін». Після цього зміни можуть бути надіслані на віддалений репозиторій за допомогою git push;

Робота з гілками – створення нових гілок (git branch <назва>) і перехід між ними (git

checkout <назва>).

Для ефективної командної роботи важливо застосовувати чітку стратегію гілкування, яка дозволить уникати конфліктів та забезпечить безперервний розвиток проекту. Зазвичай використовуються такі стратегії, як Git Flow або GitHub Flow, які передбачають наявність основних гілок для розробки, тестування та випуску нових версій проекту.

3 РЕАЛІЗАЦІЯ ГРИ «TANKI 3D»

3.1 Проєктування та реалізація основних механік

У процесі розробки механізму руху танків у грі я використав компонент Rigidbody для фізичної взаємодії з навколишнім середовищем, а також створив клас TankController, який відповідає за обробку введення від користувача та управління рухом танка. Для підтримки двох гравців я реалізував окремі схеми керування для кожного танка, що забезпечує інтуїтивно зрозуміле управління кожним танком незалежно один від одного. Налаштування компонента Rigidbody я оптимізував для балансу між реалістичністю та геймплейною зручністю. Зокрема, обмеження обертання (Freeze Rotation) запобігають небажаним перекиданням танка, а Continuous Collision Detection забезпечує точність зіткнень снарядів. На рисунку 3.1 представлена конфігурація компонента Rigidbody для танка в Unity Inspector.

Рисунок 3.1 – Налаштування Rigidbody танка

Для досягнення реалістичної фізики в грі використовуються кілька критичних параметрів. Маса об’єкта (Mass = 1) визначає його інерцію – здатність чинити опір зміні швидкості. Гальмівна сила (Drag та Angular Drag = 1) відповідає за опір поступальному та обертальному руху, що дозволяє зробити керування танком більш стабільним. Обмеження по осях (Freeze Position / Rotation) застосовуються для запобігання небажаному зсуву або перекиданню об’єкта. Також важливо використовувати режим виявлення зіткнень Collision Detection (Continuous), який забезпечує точну обробку швидких об’єктів, таких як снаряди.

Керування рухом танка реалізовано за допомогою кількох методів. Метод HandleMovement() відповідає за обробку вводу користувача, визначаючи, чи повинен танк рухатися вперед, назад або повертатись. Метод MoveTank() реалізує безпосереднє переміщення і обертання танка на сцені відповідно до введених команд. Щоб уникнути небажаного накопичення інерції, яке може спричинити неконтрольований рух після відпускання клавіш, використовується метод ResetInertia() – він скидає залишкову швидкість, забезпечуючи точність та стабільність керування. На рисунку 3.2 представлена структура GameObject танка в Unity Editor.

Рисунок 3.2 – Ієрархія об'єкта Tank1

Об'єкт танка включає кілька ключових компонентів, необхідних для його повноцінного функціонування в грі. Компонент TankCollider1 виступає як колайдер, який забезпечує фізичну взаємодію танка з оточенням, зокрема зі стінами, снарядами та іншими об'єктами. Точка firePoint використовується як місце появи снарядів і безпосередньо пов’язана з методом Shoot(), який відповідає за стрільбу. Для створення атмосферності, особливо у зимовій тематиці, додається об’єкт WinterEF, що містить візуальні ефекти, наприклад, частинки снігу. Нарешті, shootpr є префабом снаряда, який інстанціюється під час стрільби, забезпечуючи механіку атаки у грі.

Метод HandleMovement() є основним елементом, що обробляє ввід від користувача для визначення напрямку руху та повороту танка. В залежності від того, який танк (перший чи другий) виконує дію, використовуються різні осі вводу.

```
void HandleMovement(){

float move=0;

float rotate=0;

if(gameObject.name==«Tank1»){

move=Input.GetAxis(«Vertical1»)*moveSpeed;

rotate=Input.GetAxis(«Horizontal1»)*rotateSpeed;}

else if(gameObject.name==«Tank2»){

move=Input.GetAxis(«Vertical2»)*moveSpeed;

rotate=Input.GetAxis(«Horizontal2»)*rotateSpeed;}

MoveTank(move,rotate);

if(trailParticles!=null){

if(Mathf.Abs(move)>0.1f)

{if(!trailParticles.isPlaying)

trailParticles.Play();

}else{

if(trailParticles.isPlaying)
```

trailParticles.Stop();}}}

У даному методі я реалізував перевірку, який танк зараз активний (Tank1 або Tank2), після чого для кожного танка здійснюється зчитування введення з осей «Vertical1», «Horizontal1» або «Vertical2», «Horizontal2», в залежності від гравця. Ці осі визначають напрямок руху вперед/назад і поворот танка. Після цього значення передаються до методу MoveTank(), що відповідає за фактичне переміщення та обертання танка.

Для реалізації мультиплеєрного керування я налаштував окремі осі введення в Unity Input Manager. Кожен танк використовує власні параметри Horizontal/Vertical (наприклад, Horizontal1 для танка гравця 1), що дозволяє призначати різні клавіші або геймпадні кнопки. Налаштування Sensitivity та Gravity забезпечують плавність руху, а Type визначає джерело введення.

Крім того, цей метод також реалізує управління ефектами слідів від гусениць танка. Якщо швидкість танка перевищує певний поріг (0.1), активується відтворення часток, що імітують сліди від гусениць. На рисунку 3.3 представлена конфігурація осей введення для двох гравців у Unity Input Manager.

Рисунок 3.3 – Конфігурація Input Manager

Для керування рухом і стрільбою в грі використовуються параметри введення, що визначають поведінку елементів Horizontal, Vertical та Fire. Зокрема, задається призначення клавіш, наприклад, клавіші «A» і «D» використовуються для осі Horizontal1, що відповідає за поворот танка вліво та вправо. Параметр чутливості (Sensitivity = 3) 7 визначає, наскільки швидко система реагує на натиснення – вищі значення забезпечують більш різке реагування. Значення гравітації (Gravity = 3) визначає швидкість, з якою значення осі повертається до нуля після відпускання клавіш, що забезпечує плавність руху. Нарешті, тип введення задається як «Key or Mouse Button», що означає використання клавіш або миші для активації відповідної дії. На рисунку 3.4 представлена блок-схема руху танку.

7 Рисунок 3.4 – Блок-схема руху танку

Метод MoveTank() відповідає за фактичне переміщення танка. Я використовую фізичні методи компонента Rigidbody, щоб отримати точний контроль над рухом і гарну інтеграцію з фізикою гри. Для плавності руху я застосував методи MovePosition() і MoveRotation(). Вони дозволяють переміщати та повертати танк без використання прямої сили, що допомагає уникнути проблем з колізіями і непередбачуваними рухами.

void MoveTank(float move, float rotate){

// Переміщення танка

if (move != 0){

Vector3 movement = transform.forward * move * Time.fixedDeltaTime;

rb.MovePosition(rb.position + movement);}

// Обертання танка

if (rotate != 0){

Quaternion turnRotation = Quaternion.Euler(0f, rotate * Time.fixedDeltaTime, 0f);

rb.MoveRotation(rb.rotation * turnRotation);}}

Якщо move не дорівнює нулю, танк рухається вперед або назад. Я обчислюю вектор руху на основі орієнтації танка (transform.forward), множу його на швидкість і Time.fixedDeltaTime, щоб компенсувати змінну кількість кадрів. Застосовую rb.MovePosition(), щоб забезпечити плавне переміщення з урахуванням фізичних взаємодій.

Якщо rotate не дорівнює нулю, танк обертається навколо осі Y. Я створюю кватерніон turnRotation з кутом, пропорційним значенню rotate і Time.fixedDeltaTime, а потім застосовую rb.MoveRotation() для обертання танка.

Цей метод допомагає контролювати інерцію танка. Якщо нічого не робити, танк може продовжувати рух або обертання після того, як я перестав ним керувати, через накопичену інерцію. Щоб уникнути непередбачуваної фізики, я скидаю лінійну (velocity) і кутову (angularVelocity) швидкості, що дозволяє танку повністю зупинитися.

void ResetInertia(){

rb.velocity = Vector3.zero; // Скидання лінійної швидкості

rb.angularVelocity = Vector3.zero; // Скидання кутової швидкості

Команда rb.velocity = Vector3.zero скидає лінійну швидкість, повністю зупиняючи рух танка в усіх напрямках. Аналогічно, rb.angularVelocity = Vector3.zero скидає кутову швидкість, припиняючи будь–яке обертання танка. Застосування цих команд є важливим для того, щоб уникнути небажаного руху або обертання, коли користувач більше не подає керуючих команд. Це дозволяє досягти точного й передбачуваного керування без інерції, що особливо важливо для ігор з миттєвою реакцією на натискання клавіш.

Я реалізував систему пострілів з обмеженою кількістю снарядів і автоматичним

перезаряджанням. Взаємодія снарядів із танками і стінами додає грі динаміки і реалістичності. Для цього використав такі основні компоненти: клас TankController для керування пострілами, клас Bullet – для логіки снарядів, і корутину Reload() для перезаряджання.

Основні компоненти системи пострілів:

Клас TankController – відповідає за керування пострілами танка;

Клас Bullet – реалізує поведінку снарядів, включаючи рикошети та взаємодії з іншими об'єктами;

Корутина Reload() – реалізує перезаряджання снарядів з певним інтервалом.

Метод Shoot() відповідає за процес пострілу. При його виклику відбувається перевірка наявності снарядів і можливість стріляти. Якщо снаряди є, виконується кілька дій:

Перш ніж стріляти, перевіряється, чи є снаряди в наявності та чи можна зробити постріл (умова canShoot);

Якщо об'єкт muzzleFlash не дорівнює нулю, запускається анімація спалаху;

Інстанціюється новий об'єкт снаряда, який запускається у напрямку гармати танка з певною швидкістю;

Відтворюється звук пострілу;

Снаряд знищується через 10 секунд після того, як він був випущений, для запобігання його безкінечному існуванню в грі;

Після кожного пострілу оновлюється кількість снарядів в UI.

void Shoot(){

if (currentBullets > 0 && canShoot){

canShoot = false;

StartCoroutine(ShootCooldown());

if (muzzleFlash != null){

muzzleFlash.Play(); }

GameObject bullet = Instantiate(bulletPrefab, firePoint.position, firePoint.rotation);

```
Rigidbody rbBullet = bullet.GetComponent<Rigidbody>();

rbBullet.velocity = firePoint.forward * bulletSpeed;

if (audioSource != null && shootSound != null){

audioSource.clip = shootSound;

audioSource.outputAudioMixerGroup =
gameAudioMixer.FindMatchingGroups(«ShootGroup»)[0];

audioSource.Play();}

Destroy(bullet, 10f);

currentBullets––;

UpdateBulletCountUI();}}
```

Корутина Reload() відповідає за автоматичне поповнення боєзапасу снарядами. Вона працює в циклі, постійно перевіряючи, чи є місце для нових снарядів. Якщо поточна кількість снарядів менша за максимально допустиму, запускається процес перезаряджання з інтервалом у 6 секунд. Після кожного поповнення боєзапасу оновлюється інтерфейс користувача, щоб відобразити актуальну кількість доступних снарядів.

```
IEnumerator Reload(){

while (true){

if (currentBullets < maxBullets && !isReloading){

isReloading = true;

yield return new WaitForSeconds(6f); // Перезарядка триває 6 секунд

currentBullets++;

UpdateBulletCountUI();

isReloading = false;}

yield return null;}}
```

Клас Bullet відповідає за поведінку снарядів після їх випуску, включаючи зіткнення з іншими об'єктами, рикошети та взаємодію з танками або стінами. На рисунку 3.5

представлені компоненти об'єкта снаряда в Unity Inspector.

Рисунок 3.5 – Компоненти об'єкта снаряда в Unity Inspector.

Нижче наведено Bullet (Script):

Bullet Speed – швидкість польоту (відповідає параметру bulletSpeed у коді);

Explosion Prefab – префаб для ефекту вибуху;

Ricochet Sound – звук рикошету (використовується в OnCollisionEnter()).

Нижче наведено Capsule Collider:

Радіус (Radius = 0.5) і висота (Height = 1) для точної детекції зіткнень.

Обробка зіткнень (метод OnCollisionEnter):

Якщо снаряд потрапляє в танк (позначений тегом «Player»), створюється вибух, відтворюється відповідний звук і знищується об'єкт снаряда та танк. Окрім того, збільшується рахунок для відповідного гравця;

Якщо снаряд потрапляє в стіну, відтворюється звук рикошету.

void OnCollisionEnter(Collision collision){

if (collision.gameObject.CompareTag(«Player»)){

if (explosionPrefab != null){

Instantiate(explosionPrefab, collision.transform.position, Quaternion.identity);}

PlaySound(explosionSound, explosionMixerGroup, collision.transform.position);

if (collision.gameObject.name == «Tank1»){

ScoreManager.IncrementTank2Score();}

else if (collision.gameObject.name == «Tank2»){

ScoreManager.IncrementTank1Score();}

Destroy(collision.gameObject);

Destroy(gameObject);}

else if (collision.gameObject.CompareTag(«Wall»)){

PlaySound(ricochetSound, ricochetMixerGroup, transform.position);}}

Як видно на Рис. 3.4, поведінка снаряда контролюється через компонент Bullet, який використовує фізичний матеріал з відскоком (Bounciness). Це забезпечує реалістичні рикошети, описані в методі OnCollisionEnter().

Метод FixedUpdate() використовую для забезпечення стабільної швидкості снаряда, незалежно від того, чи відбулося зіткнення з іншими об'єктами. У ньому я перевіряю швидкість снаряда, і якщо вона відрізняється від бажаної, нормалізую швидкість.

void FixedUpdate(){

if (rb.velocity.magnitude != bulletSpeed){

rb.velocity = rb.velocity.normalized * bulletSpeed;}}

Взаємодія об'єктів у грі реалізована через вбудований фізичний рушій Unity, який використовує компоненти Rigidbody та Collider. Це дозволяє створити реалістичну фізику руху і зіткнень об'єктів, таких як танки та снаряди. Основною метою було забезпечити правдоподібну фізику руху, взаємодії з іншими об'єктами, а також додати ефекти вибухів та рикошетів для покращення ігрового досвіду. На рисунку 3.6 представлено конфігурація Physic Material для реалізації механіки рикошету снарядів.

Рисунок 3.6 – Налаштування фізичного матеріалу «Trick Shot» снаряда

Ключові параметри фізики в грі забезпечують реалістичну поведінку об'єктів під час руху, зіткнень та рикошетів. Параметр Dynamic Friction (динамічне тертя) визначає опір руху об’єкта під час ковзання по поверхні. Для снарядів часто задається низьке значення (наприклад, 0), щоб мінімізувати гальмування. Static Friction (статичне тертя) відповідає за опір при початку руху та зазвичай встановлюється на рівні, близькому до динамічного тертя, для узгодженої поведінки.

Параметр Bounciness (відскік) є ключовим для реалізації рикошетів – він визначає, яку частину швидкості збереже снаряд після удару об інші об’єкти. Значення цього параметра напряму впливає на те, наскільки сильно і як часто снаряд буде відскакувати. На рисунку 3.7 представлена блок-схема стрільбу та фізика снарядів.

Рисунок 3.7 – Блок-схема стрільби та фізики снарядів

Для регулювання взаємодії між поверхнями використовуються режими об’єднання властивостей. Friction Combine (об’єднання тертя) часто встановлюється в режим «Average», що забезпечує усереднене значення тертя між двома об'єктами. Інші

можливі варіанти включають Multiply, Minimum та Maximum – вони змінюють спосіб комбінування тертя. Аналогічно, Bounce Combine (об’єднання відскоку) також встановлюється в «Average», що забезпечує стабільну й передбачувану поведінку при зіткненнях.

Основні елементи фізичної взаємодії в ігровому процесі реалізуються через кілька важливих механізмів. Метод OnCollisionEnter() використовується для виявлення зіткнень між об'єктами та обробки наслідків таких взаємодій, наприклад, зменшення здоров’я чи запуску ефектів. Керування швидкістю снарядів реалізовано таким чином, щоб забезпечити сталість швидкості незалежно від частоти кадрів, що важливо для точності геймплею. При зіткненні снаряда з танком можуть створюватися візуальні та звукові ефекти вибуху, які підсилюють реалістичність гри та надають гравцеві зворотний зв’язок.

Важливою частиною фізичної взаємодії є обробка зіткнень між різними об'єктами (танками, стінами, снарядами). Для цього використовую компоненти Collider та метод OnCollisionEnter(). Кожного разу, коли снаряд або танк стикається з об'єктом, я визначаю тип зіткнення і виконую відповідні дії.

Наприклад, при зіткненні снаряда з танком створюється ефект вибуху, знищується танк і снаряд, а також збільшується рахунок гравця.

```
void OnCollisionEnter(Collision collision){

if (collision.gameObject.CompareTag(«Player»)){

// Створення вибуху при зіткненні з танком

if (explosionPrefab != null){

Instantiate(explosionPrefab, collision.transform.position, Quaternion.identity);}

PlaySound(explosionSound, explosionMixerGroup, collision.transform.position);

if (collision.gameObject.name == «Tank1»){

ScoreManager.IncrementTank2Score();}

else if (collision.gameObject.name == «Tank2»){

ScoreManager.IncrementTank1Score();}

Destroy(collision.gameObject);

Destroy(gameObject);}
```

else if (collision.gameObject.CompareTag(«Wall»)){

// Звук рикошету при зіткненні зі стіною

PlaySound(ricochetSound, ricochetMixerGroup, transform.position);

На рисунку 3.8 представлена демонстрація механіки рикошетів.

Рисунок 3.8 – Демонстрація механіки рикошетів

Для стабільної фізики снарядів, яка не залежить від частоти кадрів, я використовую метод FixedUpdate(). Він дозволяє коректно оновлювати швидкість снарядів незалежно від кількості відрендерених кадрів.

Зокрема, в методі FixedUpdate() перевіряється швидкість снаряда, і якщо вона відрізняється від бажаної, швидкість нормалізується до значення bulletSpeed. Це забезпечує стабільну поведінку снарядів в грі.

void FixedUpdate(){

if (rb.velocity.magnitude != bulletSpeed){

rb.velocity = rb.velocity.normalized * bulletSpeed;

}

}Програмні скрипти гри наведено в Додатку А.

Як видно снаряди зберігають швидкість після рикошету (контролюється параметром bulletSpeed у методі FixedUpdate()), а їх траєкторія точно враховує нормаль поверхні. На рисунку 3.9 представлена блок-схема система очків.

Рисунок 3.9 – Блок-схема система очків

Для поліпшення ігрового досвіду додається ефект звуку, який супроводжує різні події, такі як постріли, вибухи та рикошети. Всі звукові ефекти відтворюються через спеціальний метод PlaySound(), який створює новий об'єкт звуку, відтворює його і знищує після завершення звучання.

Метод PlaySound() викликається при зіткненні снаряда з танком або стіною, забезпечуючи відповідний аудіовізуальний досвід для гравця.

private void PlaySound(AudioClip clip, AudioMixerGroup mixerGroup, Vector3 position){

if (clip == null) return;

GameObject soundObject = new GameObject(«SoundObject»);

AudioSource audioSource = soundObject.AddComponent<AudioSource>();

audioSource.clip = clip;

audioSource.outputAudioMixerGroup = mixerGroup;

audioSource.spatialBlend = 1.0f; // Для 3D звуку

audioSource.Play();

Destroy(soundObject, clip.length); // Знищення об'єкта після завершення звуку

Аудіокліп clip — це звук, який буде відтворюватися під час певної події. mixerGroup визначає, через який аудіо мікшер проходитиме цей звук, що дозволяє точно налаштовувати гучність і напрямок звучання в загальній мікшерній системі. position вказує просторову точку, де має бути відтворений звук — зазвичай це координати об’єкта, з яким сталося зіткнення або інша взаємодія.

Одним з важливих елементів фізики є рикошети снарядів. Коли снаряд потрапляє у стіну або іншу тверду поверхню, його траєкторія змінюється відповідно до принципів фізики. У нашій реалізації рикошет супроводжується відповідним звуковим ефектом, що надає грі додаткової реалістичності.

else if (collision.gameObject.CompareTag(«Wall»)){

// Відтворення звуку рикошету при зіткненні з стіною

PlaySound(ricochetSound, ricochetMixerGroup, transform.position);}

3.2 Розробка інтерфейсу та локалізація

Для забезпечення інтерактивності та зручності гри я розробив кілька ключових елементів інтерфейсу користувача (UI), які надають гравцеві важливу інформацію про хід гри. Ці елементи включають відображення рахунку для кожного гравця, кількості доступних снарядів, меню паузи, налаштування графіки та звуку, а також зворотний відлік на початку раунду.

Всі елементи UI я реалізував за допомогою компонентів TextMeshPro для тексту високої якості та Canvas для позиціонування і масштабування UI елементів. Використання TextMeshPro забезпечує чіткість та зручність читання тексту в грі, а Canvas дозволяє ефективно організувати розташування різних елементів на екрані. На рисунку 3.10 представлено конфігурацію основних компонентів інтерфейсу.

Рисунок 3.10 – Налаштування UI–елементів у Unity

Основні компоненти інтерфейсу:

Рахунок для кожного гравця – показує поточний рахунок гравця, що дозволяє стежити за результатами;

Кількість доступних снарядів – інформує гравця про залишок снарядів та потребу в перезаряджанні;

Меню паузи – надає гравцю можливість призупинити гру для налаштувань або виходу.

Налаштування графіки та звуку – дозволяє гравцеві налаштовувати графіку та звукові ефекти під свої переваги;

Зворотний відлік – відображає час до початку раунду, щоб створити атмосферу напруження.

На рисунку 3.11 представлено головний інтерфейс гри під час бою.

Рисунок 3.11 – Ігровий інтерфейс

Інформаційні панелі в інтерфейсі гри відіграють ключову роль у наданні гравцю актуальних даних під час ігрового процесу. У верхній частині екрана розміщується лічильник рахунку, який відображає поточні очки або кількість знищених супротивників. Це дає змогу гравцям слідкувати за своїм прогресом і змагатися між собою.

У нижніх кутах екрана – ліворуч і праворуч – розміщуються панелі відображення боєприпасів для кожного гравця. Вони показують кількість доступних снарядів у поточному магазині, дозволяючи оцінити, коли потрібно перезаряджатись.

До інтерактивних елементів інтерфейсу належить кнопка паузи, яка розміщується в нижній частині екрана. Вона дозволяє призупинити гру, відкривши додаткове меню або тимчасово зупинивши ігровий процес, що особливо важливо для казуального геймплею та можливості перепочинку.

Для забезпечення динамічного оновлення UI я зробив функції, які автоматично оновлюють текстові значення, що відображають рахунок і кількість снарядів.

Метод UpdateScoreUI() відповідає за оновлення відображення поточного рахунку гравців. Кожного разу, коли змінюється рахунок, ця функція викликається для оновлення тексту на екрані.

void UpdateScoreUI(){

if (scoreTextTank1 != null){

scoreTextTank1.text = $»{ScoreManager.Tank1Score}»; // Оновлення рахунку для Tank1}

if (scoreTextTank2 != null){

scoreTextTank2.text = $»{ScoreManager.Tank2Score}»; // Оновлення рахунку для Tank2}}

Цей метод перевіряє наявність текстових елементів для кожного танка (scoreTextTank1 і scoreTextTank2) і оновлює їх відповідно до поточного рахунку, який зберігається у ScoreManager.Tank1Score та ScoreManager.Tank2Score.

Для відображення кількості доступних снарядів я створив метод UpdateBulletCountUI(). Він отримує локалізований текст для відображення кількості снарядів, що дозволяє легко адаптувати інтерфейс до різних мов.

void UpdateBulletCountUI(){

if (bulletCountText != null){

string localizedText = GetLocalizedString(«BULLETS_COUNT»); // Отримуємо локалізований рядок

bulletCountText.text = string.Format(localizedText, currentBullets); // Оновлюємо текст

Цей метод перевіряє, чи є елемент bulletCountText для відображення кількості снарядів, а потім оновлює текст, замінюючи шаблонне місце для числа на фактичну кількість снарядів, що зберігається в currentBullets. Локалізація дозволяє забезпечити правильний переклад тексту для різних мов.

Для зручності локалізації я використовую метод GetLocalizedString(), який отримує відповідні текстові рядки для різних мов. Це допомагає підтримувати кілька мов і легко адаптувати гру до різних культур.

string GetLocalizedString(string key){

// Псевдокод для отримання локалізованого рядка за ключем

return LocalizationManager.Instance.GetString(key);}

Таким чином, локалізація стає важливою частиною гри, адже дозволяє забезпечити універсальність ігрового інтерфейсу для міжнародної аудиторії.

Для підтримки мультимовності я використав систему локалізації Unity, що дозволяє обирати між різними мовами. У проєкті підтримуються українська та англійська. Всі текстові рядки зберігаються в окремих таблицях локалізації, що спрощує додавання нових мов і роботу з текстовими елементами. На рисунку 3.12 представлено таблицю локалізації для інтерфейсу гри.

Рисунок 3.12 – Налаштування локалізації тексту в Unity

Основні параметри налаштування локалізації включають вибір поточної таблиці та режим завантаження. Поточна таблиця має назву UL_TEXT і належить до типу StringTable у складі StringTable Collection. Встановлений режим роботи – Preload, що означає попереднє завантаження всіх текстових записів до початку сцени, забезпечуючи швидкий доступ до них під час гри без затримок.

Технічні деталі конфігурації таблиці також мають значення для оптимізації роботи. Вказано розмір сторінки – 50 записів, що дозволяє ефективно управляти великими обсягами текстових рядків. Крім того, система локалізації підтримує дві мови – англійську (en) та українську (uk), що забезпечує зручність для різних груп користувачів та адаптацію гри під міжнародну аудиторію.

Система локалізації дозволяє легко додавати нові мови та оновлювати тексти без змін коду. Наявність української локалізації покращує доступність гри для україномовних гравців.

Основні компоненти локалізації:

Таблиці локалізації – для кожної мови створено окремі таблиці, де зберігаються відповідні переклади текстових рядків (рис. 3.9);

Функція GetLocalizedString() – дозволяє отримувати локалізовані рядки за їхнім ключем;

Слухач події зміни мови LocalizationSettings.SelectedLocaleChanged – забезпечує відстеження зміни мови гри та автоматичне оновлення UI елементів.

Реалізація локалізації. Метод OnLocaleChanged() реагує на зміну вибраної мови і викликає функцію UpdateAllButtonTexts(), щоб оновити текст на всіх елементах, де використовується локалізований текст. Це забезпечує динамічну зміну інтерфейсу при зміні мови без необхідності перезапуску гри.

private void OnLocaleChanged(Locale locale){

UpdateAllButtonTexts(); // Оновлення текстів всіх кнопок при зміні мови}

Метод UpdateAllButtonTexts() викликає окремі функції для оновлення тексту кожної

кнопки або елемента UI, який використовує локалізований текст. Це дає змогу централізовано оновлювати всі необхідні елементи інтерфейсу при зміні мови.

private void UpdateAllButtonTexts(){

UpdateSoundButtonText();

UpdateShadowButtonText();

UpdateAntiAliasingButtonText();

UpdateLanguageButtonText();}

Функція GetLocalizedString() є основним способом отримання локалізованих рядків за їх ключами. Ключі використовуються для зберігання тексту в таблицях локалізації, і функція повертає відповідний переклад залежно від вибраної мови.

private string GetLocalizedString(string key){

Return LocalizationSettings.StringDatabase.GetLocalizedString(tableReference, key);

// Отримує локалізований текст з таблиці за ключем}

Ця функція дозволяє звертатися до бази даних локалізованих рядків, де кожен текст зберігається під певним ключем. Наприклад, рядок для відображення кількості снарядів або кнопок меню.

Для перемикання між мовами я створив функцію ToggleLanguage(). Вона перевіряє, яка мова зараз вибрана (українська або англійська), і змінює її на протилежну. Також мова зберігається в PlayerPrefs, щоб при наступному запуску гри зберігався останній вибір мови.

public void ToggleLanguage(){

isUkrainian = !isUkrainian; // Перемикаємо значення мови

int languageIndex = isUkrainian ? 1 : 0; // 1 – українська, 0 – англійська

// Задаємо вибрану локаль для гри

LocalizationSettings.SelectedLocale
=LocalizationSettings.AvailableLocales.Locales[languageIndex];

PlayerPrefs.SetInt(«SelectedLanguage», languageIndex); // Зберігаємо вибір мови в PlayerPrefs

PlayerPrefs.Save(); // Зберігаємо налаштування}

Ця функція дає гравцеві змогу змінювати мову під час гри, і вибір мови зберігається між сесіями, що дає зручність при повторному запуску гри.

3.3 Аудіовізуальні ефекти

Для покращення ігрового досвіду та занурення в атмосферу я реалізував різноманітні звукові та візуальні ефекти, які додають реалістичності та динамізму до гри. Звуковий супровід і візуальні ефекти працюють разом, щоб створити більш живе і динамічне середовище. У цій секції описано, як я реалізував аудіо– та візуальні ефекти в грі.

Звукові ефекти в грі важливі для створення атмосфери та забезпечення відчуття реалістичності. Для цього я використав систему AudioMixer, що дозволяє ефективно керувати різними групами звуків та налаштовувати їх параметри для досягнення бажаного ефекту. На рисунку 3.13 представлена структура звукових груп у Unity Audio Mixer.

Рисунок 3.13 – Структура звукових груп у Unity Audio Mixer

Звукові ефекти в грі згруповані за категоріями для більш точного керування їхнім звучанням та обробкою. До таких груп належать:

ShootGroup – відповідає за звуки пострілів танків; у цій групі налаштовано компресію, щоб збалансувати гучність незалежно від частоти вогню;

ExplosionGroup – містить звуки вибухів, які мають ефект реверберації, що додає їм просторового звучання;

RicochetGroup – відповідає за рикошети; у ній використовується обрізання високих частот для зменшення різкості звуку.

Загальний діапазон гучності звуків варіюється від +20 dB (максимально) до –80 dB (мінімально), що дозволяє точно налаштувати баланс між різними звуками. Крім того, використовується параметр Attenuation – обмеження гучності на рівні 80 dB, щоб уникнути звукових спотворень при перевищенні пікових значень.

У грі я використовую об'єкти AudioSource, щоб відтворювати звуки, коли відбуваються певні події, такі як постріл чи вибух. Для цього я створюю тимчасові об'єкти SoundObject, які відповідають за відтворення звуку.

public void PlaySound(){

if (audioSource != null){

audioSource.Play(); // Відтворення звуку}}

Щоб зробити звукові ефекти більш реалістичними, для пострілів та вибухів я використовую просторове відтворення звуків. Це дозволяє звукам звучати в залежності від їхнього положення на сцені, створюючи враження, що звуки приходять з реальних напрямків.

private void PlaySound(AudioClip clip, AudioMixerGroup mixerGroup, Vector3 position){

if (clip == null) return;

// Створення тимчасового об'єкта для відтворення звуку

GameObject soundObject = new GameObject(«SoundObject»);

AudioSource audioSource = soundObject.AddComponent<AudioSource>();

// Налаштування параметрів звуку

audioSource.clip = clip;

audioSource.outputAudioMixerGroup = mixerGroup; // Вибір групи мікшера для звуків

audioSource.spatialBlend = 1.0f; // Просторовий звук (1.0 означає повний 3D–ефект)

audioSource.Play(); // Відтворення звуку

Destroy(soundObject, clip.length); // Знищення об'єкта після завершення відтворення

Візуальні ефекти є важливою частиною гри, оскільки вони допомагають створити атмосферу та зробити події в грі більш динамічними. Для реалізації візуальних ефектів я використовував систему ParticleSystem Unity, яка дозволяє створювати та керувати різноманітними ефектами частинок. Основні візуальні ефекти:

Спалахи при пострілах – ефекти візуальних спалахів при кожному пострілі;

Вибухи – частинки вибуху при знищенні танка;

Сліди від гусениць – частинки, що відображають сліди руху танка;

Ефекти рикошету – ефекти частинок, що з'являються при рикошетах снарядів від стін.

При кожному пострілі з танка я додаю спалах, що підсилює відчуття реалістичності. Цей ефект активується через компонент ParticleSystem.

if (muzzleFlash != null){

muzzleFlash.Play(); // Відтворення спалаху при пострілі}

Для додавання ефекту слідів від гусениць я використовував систему частинок TrailParticles, яка активується тільки при русі танка. Це допомагає візуально підкреслити рух танка і покращити зовнішній вигляд гри.

if (trailParticles != null){

if (Mathf.Abs(move) > 0.1f) // Якщо танк рухається{

if (!trailParticles.isPlaying)

trailParticles.Play(); // Активуємо частинки сліду}

else{

if (trailParticles.isPlaying)

trailParticles.Stop(); // Зупиняємо частинки, якщо танк не рухається}}

Крім спалахів, в грі присутні вибухи та рикошети при зіткненні снарядів із об'єктами. Для кожного з цих ефектів також використовуються частинки ParticleSystem. Вибухи активуються при зіткненні снаряда з танком або іншими об'єктами, а рикошети – коли снаряд відскакує від стін.

3.4 Створення ігрових рівнів

В рамках розробки ігрових рівнів для моєї гри я створив три тематичні середовища: Осінь, Зима та Весна. Кожне середовище містить свої унікальні візуальні елементи, текстури та освітлення, що створюють атмосферу для гри. На рисунку 3.14 представлена ієрархію об'єктів сцени.

Рисунок 3.14 – Структура ігрової сцени в Unity

Освітлення:

Point Light (точкові джерела світла);

1 Directional Light (напрямлене світло).

Системні об'єкти:

Main Camera (основна камера);

GameManager (керування грою);

EventSystem (обробка подій UI).

Інтерфейс:

PANELPAUSE (панель паузи);

TEXTGAMEPAUSE (текст «Пауза»);

Кнопки SETTINGS, HOME, CLOSE;

PANELSETTINGS (панель налаштувань) .

Ігрові об'єкти:

Tank1/Tank2 (танки гравців);

firePoint (точка пострілу);

shootpr (префаб снаряда);

REDTANK (префаб танку);

WinterEF (ефекти зимової тематики).

Оточення:

«Площа» (основна ігрова зона);

«Ivl 2 walls» (стіни лабіринту);

Множинні об'єкти з позначками «default» (декоративні елементи).

У всіх рівнях я використовую статичні лабіринти у вигляді стін, які формують перешкоди для руху танків. Ці стіни не рухаються, але вони виконують роль обмежувачів, через які гравці повинні маневрувати. Лабіринти додають тактичний елемент до гри, оскільки гравці повинні планувати свої рухи, щоб уникати або використовувати стіни для стратегічних маневрів. На рисунку 3.15 представлена статична стіна з якої формуються лабіринти.

Рисунок 3.15 – Статична стіна.

На кожному рівні також присутні тематичні декорації, що підсилюють атмосферу:

На рисунку 3.16 представлена тематична ігрова сцена осінь – можна побачити

моторошні дерева, гарбузи, свічки.

Рисунок 3.16 – Тематична ігрова сцена “Осінь”

На рисунку 3.17 представлена тематична ігрова сцена зима – покриті снігом дерева, новорічні подарунки, сніговики.

Рисунок 3.17 – Тематична ігрова сцена “Зима”

На рисунку 3.18 представлена тематична ігрова сцена весна – зелені дерева, камені та інші природні елементи.

Рисунок 3.18 – Тематична ігрова сцена “Весна”

Вони не впливають на ігровий процес, але додають кольору та створюють атмосферу рівня.

Вибір теми рівня та завантаження відповідних сцен відбувається через систему в MainMenuController. Після вибору гравець може насолоджуватися випадковим рівнем відповідної тематики.

public void SelectTheme(string theme){

selectedTheme = theme;

themeSelectionPanel.SetActive(false); // Закриваємо панель вибору теми

LoadRandomLevelByTheme(); // Завантажуємо рівень відповідної теми}

private void LoadRandomLevelByTheme(){

// Визначаємо масив рівнів для вибраної теми

string[] levelNames = selectedTheme.ToLower() == «halloween» ? new string[] { «halloween», «halloween 1», «halloween 2»} :

selectedTheme.ToLower() == «winter» ? new string[] { «winter», «winter 1», «winter 2»} :

selectedTheme.ToLower() == «forest» ? new string[] { «forest», «forest 1», «forest 2»} :

new string[0];

if (levelNames.Length > 0){

// Завантажуємо випадковий рівень із списку

string randomLevel = levelNames[Random.Range(0, levelNames.Length)];

SceneManager.LoadScene(randomLevel);}}

Цей код забезпечує завантаження випадкового рівня з відповідної тематики, що дає гравцеві новий досвід кожного разу.

Основними перешкодами на рівнях є статичні стіни у вигляді лабіринтів. Вони створюють складність у маневруванні та змушують гравців планувати свої рухи, щоб уникати зіткнень або використовувати стіни для покращення позиції під час бою. На рисунку 3.19 представлені стінки рівня на кожному з рівнів є нерухомими і мають форму лабіринтів.

Рисунок 3.19 – Префаб лабіринту

Які ускладнюють пересування танків, створюючи додаткові тактичні умови для гри. Вони можуть бути розміщені таким чином, що гравцям потрібно постійно маневрувати та вибирати правильні шляхи для руху, щоб уникати попадань від снарядів. Ці стіни не мають фізичної взаємодії, окрім рикошету снарядів, які можуть відскочити від них, змінюючи напрямок.

Інші об'єкти, що з'являються на рівнях, є чисто декоративними. Наприклад, ялинки, подарунки або інші об'єкти можуть бути частиною тематики, але вони не виконують роль перешкод чи обмежень для руху. Вони служать тільки для візуального ефекту та підвищення атмосферності. На рисунку 3.20 представлені декоративні об’єкти.

Рисунок 3.20 – Тематичні об’єкти

3.5 Додаткові механіки

Для надання можливості гравцям обирати бажану тематику рівня перед початком гри, я реалізував систему вибору теми. Це дозволяє зробити кожну гру унікальною, оскільки кожна тема має свою атмосферу, декорації та механіки. На рисунку 3.21 представлено інтерфейс меню для обрання сезонної тематики ігрових рівнів.

Рисунок 3.21 – Меню вибору тематики рівнів

Компоненти включають панель вибору тематики, яка відображається на стартовому екрані. Також передбачено статичне поле для збереження вибраної теми. Реалізовано завантаження відповідного рівня, що відповідає обраному користувачем сюжету.

Код для відкриття та закриття панелі вибору теми.

public void PlayGame(){

ScoreManager.ResetScores(); // Скидаємо рахунок перед початком нової гри

themeSelectionPanel.SetActive(true); // Відкриваємо панель вибору теми

SetMainMenuButtonsActive(false); // Деактивуємо кнопки головного меню}

public void CloseThemeSelectionPanel(){

themeSelectionPanel.SetActive(false); // Закриваємо панель вибору теми

SetMainMenuButtonsActive(true); // Активуємо кнопки головного меню}

Цей код дозволяє гравцям вибрати тему перед початком гри, створюючи більш персоналізований досвід.

Для покращення користувацького досвіду я реалізував систему підказок, яка доступна через спеціальне меню. Це меню дозволяє гравцям ознайомитися з базовими керівництвами та підказками, що допомагають під час гри. На рисунку 3.22 представлено інформаційне вікно з основним керуванням та стратегічними порадами.

Рисунок 3.22 – Інтерфейс підказок з керуванням та тактикою

Код для відкриття та закриття меню підказок.

public void OpenKe(){

keMenu.SetActive(true); // Відкриваємо меню підсказок

SetMainMenuButtonsActive(false); // Деактивуємо кнопки головного меню}

public void CloseKE(){

keMenu.SetActive(false); // Закриваємо меню підсказок

SetMainMenuButtonsActive(true); // Активуємо кнопки головного меню}

Цей механізм дозволяє забезпечити легкий доступ до інструкцій у грі, що важливо для нових гравців.

Для забезпечення зручності та комфорту гри на різних пристроях, я реалізував систему налаштувань. Вона дозволяє гравцям вмикати/вимикати звук, налаштовувати тіні та згладжування (Anti–Aliasing) для оптимізації графіки відповідно до їх уподобань. На рисунку 3.23 представлено вікно налаштувань з основними параметрами гри.

Рисунок 3.23 – Меню налаштувань гри

Основні налаштування включають кілька важливих категорій. По–перше, графічні параметри: тіні увімкнено для реалістичного освітлення, а згладжування встановлено на рівні 8X для високої якості країв. По–друге, аудіо–налаштування передбачають увімкнення звуку, що забезпечує повний звуковий супровід.

Також є мовні налаштування – інтерфейс локалізовано українською мовою. Для зручності користувача передбачена кнопка «ЗАКРИТИ», яка закриває меню налаштувань.

До компонентів налаштувань належать можливості вмикання та вимикання звуку, налаштування тіней (їх включення або вимикання), а також регулювання рівня згладжування (Anti–Aliasing).

Приклад коду для цих налаштувань.

public void ToggleSound(){

isSoundOn = !isSoundOn;

PlayerPrefs.SetInt(«SoundOn», isSoundOn ? 1 : 0); // Зберігаємо налаштування

UpdateSoundButtonText(); // Оновлюємо текст кнопки

AudioListener.volume = isSoundOn ? 1 : 0; // Вмикаємо або вимикаємо звук}

public void ToggleShadows(){

areShadowsOn = !areShadowsOn;{

PlayerPrefs.SetInt(«ShadowsOn», areShadowsOn ? 1 : 0); // Зберігаємо налаштування

UpdateShadowButtonText(); // Оновлюємо текст кнопки

SetShadows(areShadowsOn); // Вмикаємо або вимикаємо тіні}

public void ToggleAntiAliasing(){

antiAliasingLevel = antiAliasingLevel switch{

0 => 2,

2 => 4,

4 => 8,

_ => 0};

PlayerPrefs.SetInt(«AntiAliasingLevel», antiAliasingLevel); // Зберігаємо рівень згладжування

UpdateAntiAliasingButtonText(); // Оновлюємо текст кнопки

SetAntiAliasing(antiAliasingLevel); // Встановлюємо рівень згладжування

}

Цей код дозволяє змінювати звукові та графічні налаштування гри в реальному часі, зберігаючи їх між сесіями.

Використовуючи PlayerPrefs, я забезпечую збереження налаштувань між сесіями гри. Це дає змогу гравцям не налаштовувати параметри щоразу при запуску гри.

private void Start(){

isSoundOn = PlayerPrefs.GetInt(«SoundOn», 1) == 1; // Зчитуємо налаштування звуку

areShadowsOn = PlayerPrefs.GetInt(«ShadowsOn», 1) == 1; // Зчитуємо налаштування тіней

antiAliasingLevel = PlayerPrefs.GetInt(«AntiAliasingLevel», 4); // Зчитуємо налаштування згладжування

isUkrainian = PlayerPrefs.GetInt(«SelectedLanguage», 0) == 1; // Зчитуємо вибір мови

LocalizationSettings.SelectedLocale = LocalizationSettings.AvailableLocales.Locales[isUkrainian ? 1 : 0]; // Встановлюємо мову

LocalizationSettings.SelectedLocaleChanged += OnLocaleChanged; // Слухаємо зміни мови

UpdateAllButtonTexts(); // Оновлюємо тексти на кнопках

AudioListener.volume = isSoundOn ? 1 : 0; // Встановлюємо рівень звуку

SetShadows(areShadowsOn); // Встановлюємо рівень тіней

SetAntiAliasing(antiAliasingLevel); // Встановлюємо рівень згладжування

Зміна тіней та згладжування через Unity системні налаштування

Налаштування тіней і згладжування змінюються через системні налаштування Unity:

private void SetShadows(bool enable){

QualitySettings.shadows = enable ? ShadowQuality.All : ShadowQuality.Disable;}

private void SetAntiAliasing(int level){

QualitySettings.antiAliasing = level;}

Ці методи дозволяють змінювати якість графіки гри для покращення її продуктивності або візуального вигляду в залежності від можливостей пристрою.

3.6 Опис та аналіз гри «TANKI3D»

У рамках проєкту було створено 3D–гру «TANKI3D» – аркадний танковий шутер для двох гравців з локальним мультиплеєром. Гра розроблена на рушії Unity з використанням мови програмування C#, що забезпечує плавний ігровий процес та високу продуктивність на різних пристроях. «TANKI3D» пропонує динамічні битви двох танків на різноманітних тематичних аренах.

Гра має змагальний режим для двох гравців, де вони керують танками на одному екрані, змагаючись за найвищий рахунок. Механіка керування є простою, але захоплюючою: перший танк керується клавішами WASD для руху та клавішею F для стрільби, а другий – стрілками для руху та клавішею RightControl для стрільби. Кожен танк має обмежену кількість боєприпасів, всього 5 зарядів, з автоматичною перезарядкою по одному пострілу кожні 6 секунд, що додає стратегічної глибини. Окрім того, снаряди можуть відбиватися від стін, створюючи непередбачувані ситуації, що вимагає від гравців тактичного мислення.

Візуальні ефекти та звуковий супровід підвищують атмосферність гри: спалахи пострілів, сліди руху танків, вибухи та аудіоефекти роблять процес ще більш захоплюючим. Технічно гра використовує модульну архітектуру коду, де кожен компонент (танки, снаряди, ігровий менеджер) реалізований як окремий клас з чітко визначеними обов'язками, що полегшує підтримку та розширення гри.

Система керування станом гри реалізована через клас `GameManager`, який контролює цикл життя гри, зокрема зворотний відлік перед початком раунду та завантаження нових рівнів після завершення поточного. Підрахунок балів забезпечується статичним класом `ScoreManager`, що централізовано керує рахунком гравців і забезпечує простий доступ до цих даних з різних частин гри. Для збереження налаштувань гравця використовуються `PlayerPrefs`, що дозволяє зберігати користувацькі налаштування між сеансами гри.

Гра підтримує багатомовність, зокрема англійську та українську мови, завдяки системі

локалізації Unity. Гравці можуть вибирати між трьома тематичними світами (Halloween, Winter, Forest), кожен з яких має по три унікальні рівні з різним дизайном та перешкодами. Інтерфейс гри спроектований з урахуванням простоти та інтуїтивності, включаючи головне меню для вибору гри, налаштувань, елементів керування та перегляду статистики, а також ігровий інтерфейс, що відображає рахунок і кількість боєприпасів.

Інтерфейс також включає меню паузи, що дозволяє призупинити гру та змінити налаштування або повернутися до головного меню. У налаштуваннях можна регулювати гучність звуку, якість графіки (тіні, згладжування) та вибір мови інтерфейсу.

Технічно в грі використовується вбудована фізична система Unity для реалістичного руху танків та снарядів, а також система частинок для створення візуальних ефектів. Звукові ефекти керуються через аудіо міксер Unity, що дозволяє регулювати різні групи звуків окремо. Всі ці технічні рішення роблять гру захоплюючою і стабільною, а модульна архітектура коду та чіткий розподіл відповідальності між компонентами забезпечують легкість у підтримці та розвитку гри.

Посилання

Це джерела виділених збігів у вашому документі. Кожен збіг позначено темно-зеленим числом, яке відповідає вказаному тут джерелу. Джерела впорядковані за схожістю — чим вищий бал, тим сильніше збіг.

#	Джерело	%
1	ela.kpi.ua	0.4%
2	jetiq.vntu.edu.ua	0.4%
3	openarchive.nure.ua	0.3%
4	rbc.ua	0.2%
5	metod.vntu.edu.ua	0.1%
6	ts2.space	0.1%
7	ela.kpi.ua	0.1%
8	znaki.fm	0.1%
9	researchgate.net	0.1%
10	ua.xdvirtualreality.com	0.1%
11	repositsc.nuczu.edu.ua	0.1%
12	unity3d.com	0.1%
13	markiza-pergola.kiev.ua	0.1%
14	confscientific.webnode.com.ua	0.1%
15	isg-konf.com	0.1%
16	dou.ua	0.1%

Plag

Дякуємо, що перевірили
свій документ за допомогою
Plag!