КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет комп'ютерних наук та кібернетики

Кафедра теорії та технології програмування

Кваліфікаційна робота

на здобуття ступеня бакалавра

за спеціальністю 122 Комп'ютерні науки

на тему:

ВИКОРИСТАННЯ ІГРОВИХ РУШІЇВ ДЛЯ РОЗРОБКИ ІГРОВИХ ДОДАТКІВ

Виконав студент 4-го курсу

Сергій МАЗАЄВ _____

Науковий керівник:

доцент, кандидат фіз.-мат. наук

Людмила ОМЕЛЬЧУК _____

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____

Роботу розглянуто й допущено до захисту на засіданні кафедри теорії та технології програмування

« 01 » червня 2022 р.,

протокол № 10

Завідувач кафедри

Микола НІКІТЧЕНКО _____

Київ – 2022

РЕФЕРАТ

Обсяг роботи 39 сторінок, 20 ілюстрацій, 25 джерел посилань, 3 таблиці.

РУШІЙ, ТЕКСТУРА, СПРАЙТ, АНІМАЦІЯ, FLIPBOOK, BLUEPRINT, CLASS, UNITY, UNREAL ENGINE, ACET, ПЛАГІН, ПЛАТФОРМЕР.

Об’єктом роботи є процес розробки ігрового додатка з використанням рушіїв та без них. Предметом роботи є ігровий рушій, призначений для розробки ігрових додатків.

Метою роботи є аналіз деяких наявних на ринку ігрових рушіїв, порівняння підходів розробки гри з використанням рушіїв та без них, а також допомога у виборі рушія для створення ігор для початківців у розробці ігор.

Методи розроблення: моделювання, декларативне програмування. Інструменти розроблення: Unreal Engine 4.

Результатом розробки є опис процесу розробки базису ігрового додатка у жанрі двовимірного платформеру з використанням ігрового рушія Unreal Engine 4, а також порівняльна характеристика розглянутих ігрових рушіїв.

Практичне значення полягає в отриманні первинної інформації про популярні рушії, що має допомогти у швидшому виборі кращого для розв'язання поставленої задачі.

У ході виконання роботи були отриманні відомості про два популярних ігрових рушія та використані на практиці під час розробки базису ігрового додатка у жанрі двовимірний платформер.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

2D – двовимірний**;**

2.5D – два с половиною виміру;

3D – тривимірний;

МП – мова програмування;

ПК – персональний комп’ютер;

AI – Artificial Intelligence або штучний інтелект;

EG – Epic Games;

IDE – Integrated Development Environment;

OpenGL – Open Graphics Library;

SFML – Simple and Fast Multimedia Library;

USD – долари Сполучених Штатів Америки;

UE – ігровий рушій Unreal Engine.

ВСТУП

Оцінка сучасного стану об’єкта розробки. На сьогодні індустрія комп’ютерних ігор є об’ємним ринком розваг. Цей ринок приходить майже у будь-яке місце, куди дістались комп’ютери як такі.

Розробка ігор «з нуля» займає велику кількість часу, через необхідність продумати всі аспекти ігрового світу. Водночас у всіх іграх є дещо спільне. Насамперед це потреба промальовувати зображення на екрані, фізика перетину об’єктів, необхідність реакції на деякі події, як, наприклад, натискання кнопок чи смерть гравця.

Зважаючи на велику кількість загальних моментів у розробці ігор та на можливість уніфікації, не бажаючи переписувати базові аспекти ігор знов і знов, програмісти створили таку сутність як «ігровий рушій», що взяв ці базові функції на себе, полишаючи на розробника лише логіку взаємодії об’єктів.

Наразі існує низка ігрових рушіїв. Це можуть бути як закриті рушії під одну конкретну гру, так і рушії у вільному доступі на яких можна створювати широкий спектр ігор.

Як приклад рушія для конкретної гри можна навести Encore Engine [1] розробленою компанією Wargaming [2] для гри World of Tanks [3]. Перевагою такого підходу є оптимальність у рамках конкретної гри. Цей рушій одночасно дозволяє видавати дуже деталізовану картинку на потужних машинах та запускатись з невеликими вимогами до графіки навіть на комп’ютерах з Windows 2000 [4].

Прикладами рушіїв у вільному доступі для широкого спектра ігор є Unreal Engine [5] та Unity [6]. Перевагами такого підходу є багатофункціональність, кросплатформність. Окремо варто зазначити, що це два найбільш популярних рушії [7], тому надалі робота буде зосереджена саме на порівнянні цих рушіїв.

Актуальність роботи та підстави для її виконання. Будь-хто, хто бажає поринути у світ розробки ігор на першому ж етапі зіштовхується з проблемою вибору технологій для їх розробки. Ключовою серед технологій є саме ігровий рушій.

Щоб наочно продемонструвати фундаментальність технологій ігрового рушія у процесі розробки ігор, у рамках даної роботи буде розроблено додаток без використання ігрових рушіїв, розроблено базис гри з використанням ігрового рушія та виконано порівняння двох підходів.

Мета і завдання роботи. Метою кваліфікаційної роботи є аналіз наявних технологій розробки ігор, а також допомога у виборі рушія для створення ігор для початківців у розробці ігор. Для досягнення цієї мети поставлено такі завдання.

- Розробка простої гри без використання ігрових рушіїв.
- Огляд наявних на ринку ігрових рушіїв.
- Реалізація простої гри з використанням розглянутих ігрових рушіїв.
- Порівняння розглянутих ігрових рушіїв.

Об'єкт, методи й засоби розроблення. Об’єктом роботи є процес розроблення певних аспектів ігрової логіки засобами наявними у рушії Unreal Engine [5] та Unity [6] та без їх використання. Предметом роботи є ігровий рушій, призначений для розробки ігрових додатків. Основу роботи склав аналіз процесу розробки ігор.

Під час розробки програмного продукту використана еволюційна модель розробки, реалізацію здійснено як на основі імперативної, так і декларативної парадигм програмування.

Можливі сфери застосування. Сфера застосування ігрових рушіїв – розробка ігор та ігрових додатків.

РОЗДІЛ 1.

ОГЛЯД НАЯВНИХ НА РИНКУ ТЕХНОЛОГІЙ РОЗРОБЛЕННЯ ІГОР

Наразі основною технологією для розробки ігор є ігровий рушій. Взагалі рушій – це серце гри. Він відповідає за анімацію, візуалізацію, фізику, звук так багато чого іншого. Від програміста вимагається лише правильно задати логіку всіх цих речей.

Деякі компанії йдуть шляхом розробки власного рушія. Наприклад рушій enCore [1] від компанії Wargaming [2] для її найвідомішого проєкту World of Tanks [3].

Проте для менших розробників, будь то одна людина, чи невелика фірма, простіше використати наявні готові рушії. Вони дають купу різних інструментів, що полегшують розробку нової гри, як досвідченим програмістам, так і новачкам. Основних таких у сучасному світі ігрових розробок два: Unreal Engine [5] та Unity3d [6]. В ході роботи буде виконано порівняння та опис недоліків та переваг кожного з них.

Проте обрані для порівняння рушії не є вичерпним списком усіх наявних технологій на сьогодні. Список існуючих рушіїв є значно ширший, тому за бажанням читач може обрати інший [8].

Потрібно зазначити, що під час вибору рушія для розробки слід звернути увагу на наступні властивості:

- Мова розробки (C++, C#, тощо).
- Платформи, що підтримуються рушієм (Window, Linux, тощо).
- Розмірність результуючого продукту (2D, 2.5D, 3D).

РОЗДІЛ 2.

РОЗРОБКА ПРОСТОГО ІГРОВОГО ЗАСТОСУНКУ БЕЗ ВИКОРИСТАННЯ ІГРОВИХ РУШІЇВ

Для розробки гри було обрано мову програмування C++. Причиною такого вибору була відносна низькорівневість, яка потребує більш детального розбору механік написання гри [9].

Програми на C++ можуть бути дуже сильно прискорені при оптимальному написанні, що є великою перевагою при роботі з важко обчислюваними операціями, до яких відноситься і робота з графікою.

Оскільки, в цьому випадку задачею була розробка ігрового застосунку без використання готових рушіїв, для роботи з графікою було використано бібліотеку Simple and Fast Multimedia Library [10]. SFML є надбудовою над OpenGL [11] і дозволяє більш просто працювати з зображенням.

Зокрема, вона дозволяє завантажувати в пам’ять текстурні зображення, робити з них прямокутні спрайти, повертати їх і промальовувати у заданій точці екрану. Також SFML має декілька приємних бонусів не пов’язаних з графікою. Ця бібліотека вміє зчитувати та подавати у зручному для обробки вигляді події. Окрім цього бібліотека SFML вміє запускати, зупиняти, сповільнювати та пришвидшувати програвання музичних треків для вашої гри. Окремо треба зазначити, що розширення треків має бути у форматі *.ogg.

Для розробки використано IDE Microsoft Visual Studio 2013 [9].

Для малювання текстурних картинок використовувався сайт pixilart.com. У порівнянні з іншими редакторами він дозволяв зручніше звертатись до окремого пікселя, що в моїх очах було перевагою через невеликі розміри текстурних картинок.

2.1. Призначення і цілі створення застосунку

2.1.1. Вимоги до застосунку

Під час розробки даного ігрового додатка без використання ігрових рушіїв було поставлено декілька цілей.

- Власноруч пройти всі етапи розробки гри.
- Навчитись розробляти та реалізувати архітектуру додатку.
- Навчитись працювати з графікою на деякому базовому рівні (позиціювання та малювання текстурних картинок на екрані)
- Створити простий штучний інтелект для ботів у грі.

Реалізований ігровий додаток повинен бути розрахований на одного гравця, без виграшного фіналу.

У головного героя повинно бути кілька різних видів зброї, які повинні бути збалансовані під певну ігрову ситуацію. Боєприпасів для зброї на початку гри повинно бути або мало, або зовсім не бути, щоб змусити гравця шукати ресурси по карті.

Самі ресурси повинні знаходитись у певних місцях, положення яких теж генерується випадковим чином. Оскільки гра може бути нескінченно довга, а карта обмежена, ресурси у місцях їх схову повинні періодично поновлюватись.

Також, на карті повинні існувати точки покращення зброї. В цих точках повинна бути можливість покращити певні характеристики зброї.

2.1.2. Вимоги до функцій, які виконуються застосунком

Розділимо гру на ігрові сцени. Таких сцен буде п’ять. Опишемо кожну сцену окремо і визначимо необхідні функції для кожної сцени. Надалі зобразимо таблицею ігрові сцени та необхідний до них функціонал (див. табл. 1).

Таблиця 1. Сцени та їх функції

№	Ігрова сцена	Візуальна складова	Функціональна складова
1.	Головне меню	Перед гравцем повинно постати чотири кнопки: 1) Завантажити гру. 2) Почати нову гру. 3) Налаштування. 4) Вихід.	Дана ігрова сцена повинна переводити гравця до інших ігрових сцен або завершити роботу програми при натисканні певних клавіш.
2.	Налаштування	Текстовий опис дії певних клавіш під час гри. Перемикач складності гри.	Можливість змінення складності наступної гри.
3.	Головна ігрова сцена	Гравець повинен бачити частину ігрового поля навколо себе та об’єкти на ньому, вибрану гравцем зброю, кількість патронів до обраної зброї, очок життя та броні, гранат та час до нової хвилі ворогів.	Гравець повинен мати можливість рухатись, стріляти, кидати гранати та взаємодіяти з навколишнім світом. Можливість поставити гру на паузу та вийти до головного меню.
4.	Мінікарта	Масштабоване зображення ігрового поля. Видимі, невидимі та не відкриті клітинки поля повинні показуватись по-різному	Гравець повинен мати можливість рухатись, стріляти, кидати гранати та взаємодіяти з навколишнім світом. Можливість поставити гру на паузу та

			вийти до головної меню ігрової сцени.
5.	Сцена покращення зброї	Кнопки по вибору зброї та властивості для покращення. Кнопка «покращити».	Динамічне покращення характеристик зброї. Можливість перейти до головної ігрової сцени.

2.2. Реалізація застосунку

Всього у грі реалізовано 29 класів у 7 920-ти рядках коду. Діаграма класів знаходиться у додатку А.

При «ручній» реалізації гри були наявні три стадії.

1. Отримання та обробка подій від користувача.
2. Виконання необхідних дій, рух усіх рухомих об’єктів у грі та перевірка на перетин тих об’єктів, що перетинатись не повинні.
3. Малювання нової картинки на екрані.

Окремо цьому передує завантаження потрібних даних до оперативної пам’яті (наприклад текстурних зображень), та окремо процес гри завершується звільненням використаної пам’яті.

2.2.1. Реалізація головного меню

Перша ігрова сцена, яку бачить гравець – це головне меню. За його реалізацію відповідають класи MainMenu та Button. Попередньо завантажуються шрифти для відображення на кнопках у головному меню, також створюються об’єкти кнопок для подальшого їх відображення.

За обробку події глобально відповідає клас головного меню, проте при події «натискання кнопки» кожна кнопка змінює свій стан самостійно. Клас меню лише надає цю подію у зручному вигляді до об’єктів кнопок.

Далі, кнопка може повідомити, що вона була натиснута. У відповідь на це клас меню або звільняє пам’ять та закриває гру, або передає керування класу налаштувань, або створює екземпляр класу гри й передає керування йому.

2.2.2. Налаштування

За цю сцену відповідають класи Settings, Button та ToggleBox. Все що коїться тут аналогічне до попередньої сцени, за єдиним винятком. Змінення ToggleBox призводить до змінення значень полів у класі GlobalVariable**.**

2.2.3. Реалізація головного ігрового процесу

Перший пункт у життєвому циклі гри – завантаження необхідних даних. За візуальну частину відповідає GraphicsManager. За попередню генерацію карти та зв’язаних з нею об’єктів відповідає EnvironmentManager. Усе інше, як і команду до завантаження даних у попередніх класів дає головний клас гри – Game.

Обробкою подій займається клас гри, проте обробку частини подій він може делегувати іншим класам. Як приклад, при спробі зібрати ресурси зі сховищ обробка цієї події делегується до EnvironmentManager який перевіряє, чи досить близько гравець до сховища; чи можна зібрати ресурси зі сховища, біля якого гравець стоїть; генерує ресурси випадковим чином та передає їх гравцю інкрементувавши потрібні змінні.

При переході від головної ігрової сцени до інших клас гри делегує класу GraphiscManager подію та останній вже вибирає необхідну сцену (інтерфейс кожної сцени реалізовані окремим класом) і передає командування малюванням їй.

Далі йде виконання дій та рух. Перше що робиться, це передається значення пройденого проміжку часу, з моменту останнього виконання дій, до усіх об’єктів, які повинні працювати через певні проміжки часу. Це може бути як зброя, яка стріляє через певні проміжки, так і генератор хвиль ворогів.

Далі йде рух усіх об’єктів. Рухомими об’єктами являються гравець, вороги, куля та граната.

Після того, як всі об’єкти були переміщенні проводиться перевірка на перетин «матеріальних» об’єктів між собою. Матеріальні об’єкти це стовбури дерев, сховища, об’єкти покращення зброї, вороги, кулі та гравець. Усі матеріальні об’єкти є прямокутниками, за винятком дерев, де матеріальним є їх стовбур (деяке коло, яке менше ніж видима на екрані крона).

Якщо один з об’єктів, які перетнулися є куля, то йде перевірка, хто є ціллю кулі (гравець чи ворог). Якщо куля влучила у ціль, то з цілі списуються очки здоров’я та/або броня.

Якщо обидва об'єкти є рухомими, то, якщо один з об’єктів є будинок чи дерево, то рухомий об’єкт переміщується горизонтально чи вертикально так, щоб уникнути перетину. Якщо зіткнулись гравець та ворог – гра закінчується, вважається, що гравець програв.

Останнім йде генерація усіх потрібних об’єктів. Такими об’єктами можуть бути кулі зі зброї гравця чи ворогів, гранати чи нові хвилі ворогів. За генерацію куль відповідають об’єкти Weapon, за генерацію гранат клас Game, а за генерацію ворогів WaveManager.

Наступним йде малювання. За малювання відповідає GraphicsManager, який запускає певний сценарій малювання, який, своєю чергою, передає команду «промалюй себе» потрібним об’єктам.

Інформація, що саме промальовувати може збиратись паралельно з ігровим процесом, як у випадку мінікарти, так і розраховуватись у сам момент малювання, як у всіх інших моментах.

Є окремі речі, які потребують опису та не вписались у вище викладену модель розповіді. Перейдемо до них.

Перше, під час генерації нової хвилі ворогів відбувається збереження прогресу даної гри. Це збереження відбувається лише при повному знищенні попередньої хвилі, та повністю переписує попереднє збереження. Таким чином в один момент ви можете мати лише одне збереження.

Друге, гру можна поставити на паузу. Для цього головний клас гри створює екземпляр класу GamePause. Останній говорить блоку руху та виконання подій нічого не робити доки не буде знята пауза, в також домальовує поверх ігрової сцени напис “Game Paused”.

Окремого опису заслуговують вороги. Але не самі вороги, а те, що змушує їх рухатись та виконувати дії. AI ботів зроблений з використанням патерну State Machine. Деякі ідеї для створення ботів запозичувались з навчальних відео та книг по розробці ігор та штучного інтелекту до них [12, 13].

2.2.4. Приклади реалізація ігрового застосунку

При «ручній» реалізації ігрового застосунку, серед іншого, необхідно було реалізувати перевірку на перетин гравця/ворога із будівлями (об’єкт для перехода до режима покращення зброї, та ангару для збору ресурсів). Сама перевірка на перетин виконана досить тривіально, тому не будемо на ній зупинятись. А от далі потрібно визначити, з якою з чотирьох сторін прямокутника перетнувся гравець, та в яку сторону його змістити, щоб уникнути ситуації перетину.

Ідея алгоритму полягає у використанні орієнтованої площі трикутника, а саме, її знаку. Взявши дві протилежні вершини прямокутника, який утворює об’єкт будівлі, ми отримаємо діагональ об’єкту. Знайшовши орієнтовану площу трикутника за вершинами, що є двома обраними кутами прямокутника та центром об’єкту гравця ми можемо знати, вище чи нижче за діагональ гравець.

Дві діагоналі, своєю чергою, поділять простір на чотири сектори, де кожна сторона прямокутника буде повністю належати одному з секторів. Отримавши знакову площу для обох діагоналей ми можемо однозначно виявити, до якого сектору належить гравець, а отже з якою зі сторін він перетнувся. Фрагменти реалізації наведено в Додатку Б.

За певний проміжок часу, який проходить між промальовуванням картинок, куля встигає подолати деяку відстань. В ідеалі, це зводиться до задачі перетину відрізку з прямокутником чи колом. Розв'язання даної задачі є досить важким для обрахунків, тому було вирішено зробити інакше.

Оскільки гра рухає об’єкти приблизно кожні 20 мілісекунд і кількість об’єктів, з якими куля може перетнутись не є досить великою, то було зроблено наступне рішення:

- розраховується відрізок, який встигла пройти куля;
- даний відрізок ділиться на 5 рівних підвідрізків;
- точка, яка відповідає кінцю цього підвідрізку, у напрямку руху кулі записується у окремий масив.

І надалі мі отримуємо задачу не перетину відрізку та фігури, а задачу перевірка належності точки фігурі, де точною є одна з п’яти точок на шляху руху кулі. Фрагменти реалізації наведено в Додатку Б.

Повний код реалізації знаходиться за посиланням на відкритий репозиторій на GitHub [14].

2.3. Використання ігрового застосунку

У інструкції користувача будуть наведені ілюстрації ігрових екранів з коротким текстовим описом та підписаними основними функціональними елементами.

1) Головне меню. На екрані гравець бачить 4 кнопки: «Нова гра», «Продовжити», «Налаштування» та «Вийти» (див. рис. 1)

Рисунок 1. Екран головного меню

2) Налаштування. Тут гравець бачить функціонал основних кнопок гри, може змінити складність гри та повернутись до головного меню (див. рис. 2)

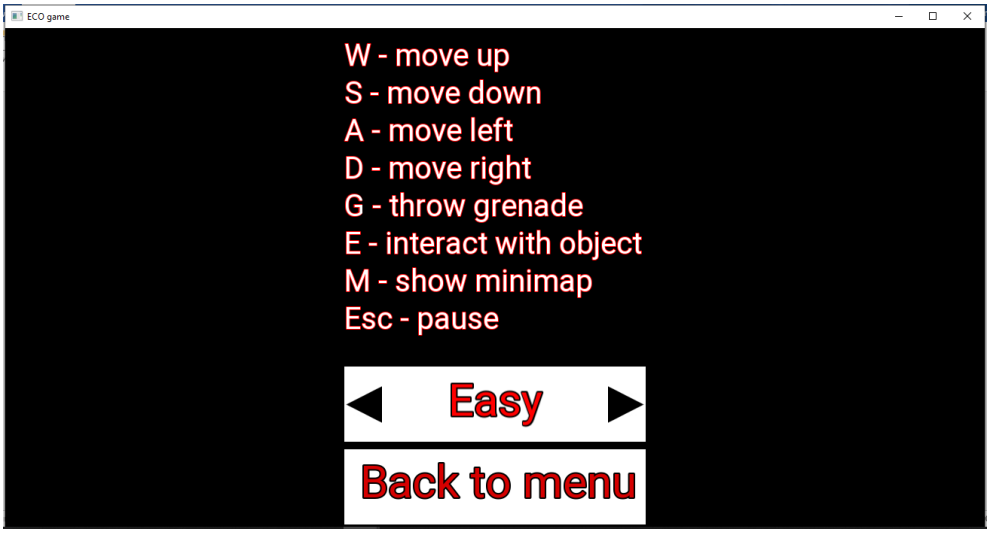


Рисунок 2. Екран налаштування

3) Сцена покращення зброї. Гравець бачить кількість очок покращення, може вибрати зброю та характеристику для покращення (див. рис. 3)

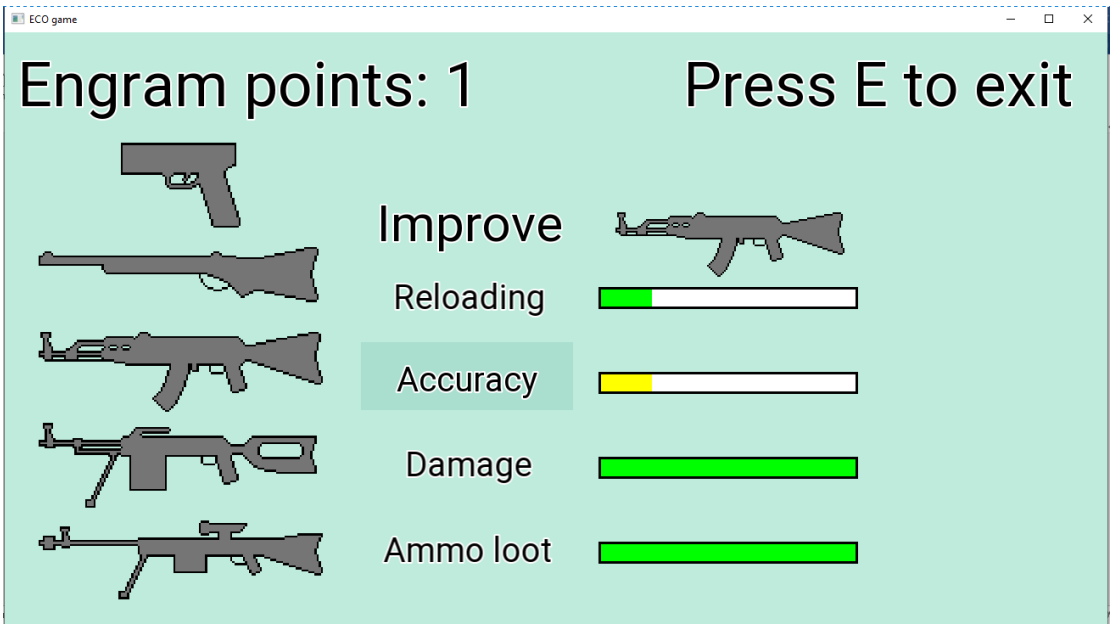


Рисунок 3. Екран покращення

4) Головна ігрова сцена. Гравець з інтерфейсу може отримати інформацію про номер наступної хвилі, час до настання наступної хвилі, логи подій, об'єкти довкола себе, кількість очок здоров'я та броні, вибрану зброю, кількість патронів до неї у магазині та в запасі та кількість гранат. (див. рис. 4).

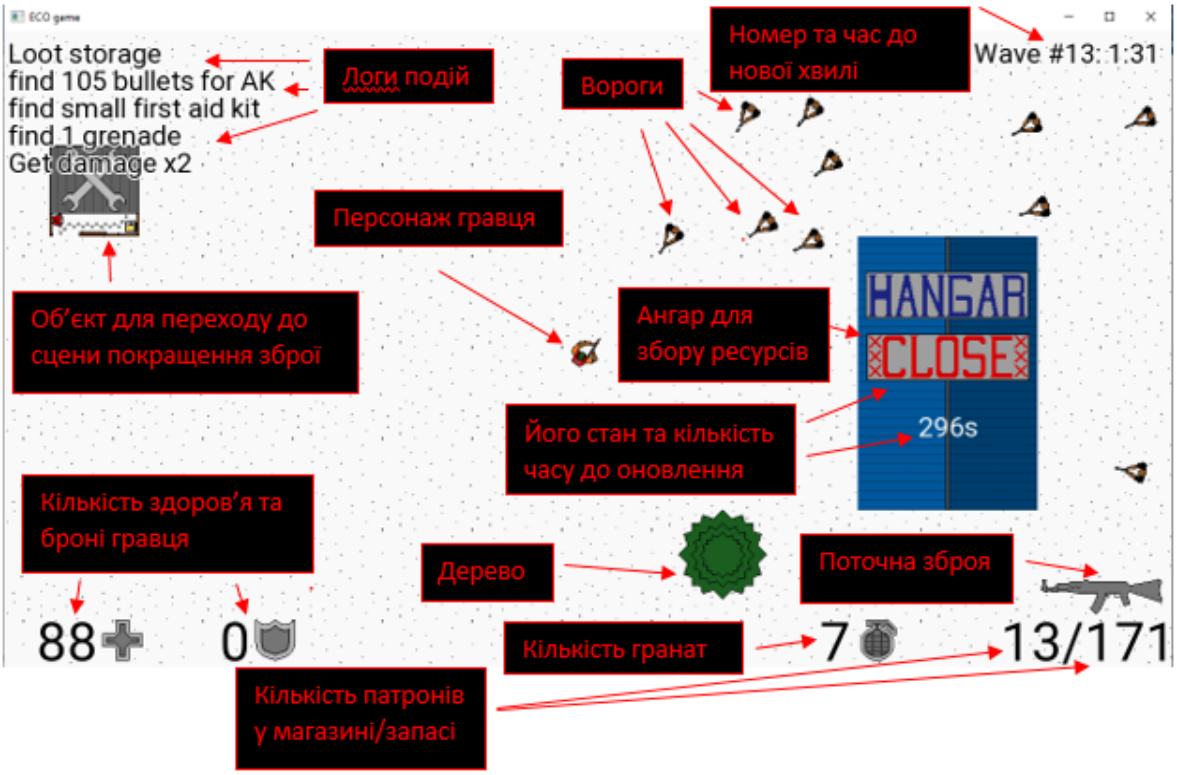


Рисунок 4. Екран головної ігрової сцени

5) Мінікарта. Схематичне зображення ігрового світу. (див. рис. 5)

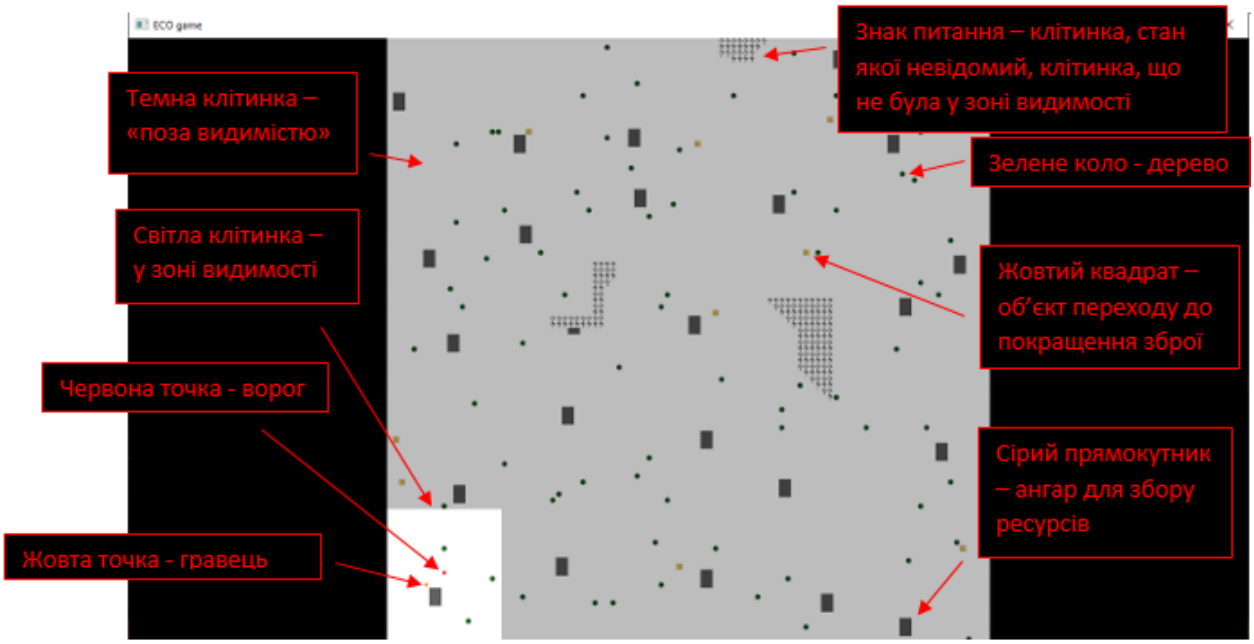


Рисунок 5. Екран мінікарти

6) Ігрова пауза. (див. мал. 6)

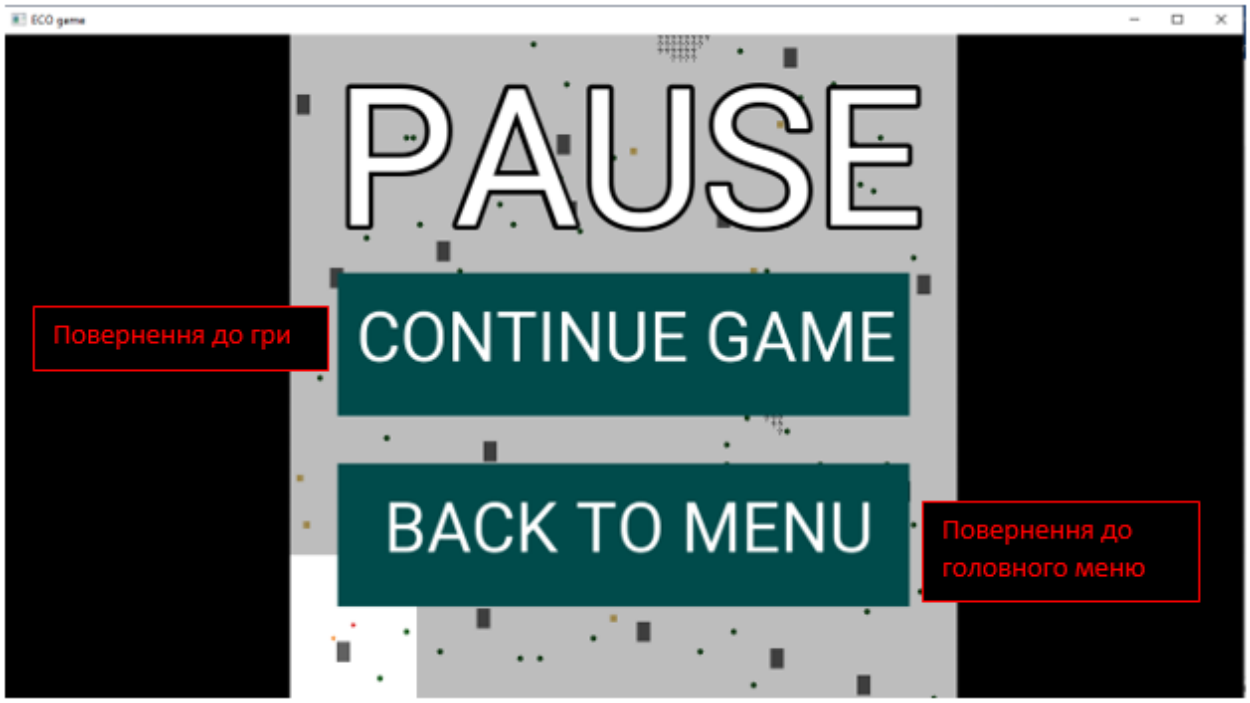


Рисунок 6. Екран паузи

2.4 Висновки за результатами розробки гри без використання рушія

Переваги: Розробка гри без використання рушіїв дає розробнику більшу владу над процесами, що виконуються у грі. Уся логіка від завантаження перших даних до момента завершення роботи програмного продукту є

зрозумілими й контрольованими. Існує можливість підібрати оптимальні методи вирішення тих чи інших задач у рамках конкретного ігрового проєкту.

Недоліки: Більшу частину розробки займає написання основ, які реалізовані у сучасних рушіях. Проблемно також оновлювати програмний застосунок. Існують складнощі з підтримкою кросплатформності. Необхідність залучати більшу кількість сторонніх бібліотек (наприклад SFML) створює ризик проблем при потенційному оновлені цих бібліотек.

РОЗДІЛ 3.
ОГЛЯД ОБРАНИХ ІГРОВИХ РУШІЇВ

3.1. Ігровий рушій Unreal Engine

На початку розглянемо кожний рушій окремо, а потім порівняємо їх. І першим до розгляду потрапляє Unreal Engine (див. рис. 7).

Рисунок 7. Логотип Unreal Engine

Опис функціоналу ігрового рушія Unreal Engine наведемо на основі буде для версії 4 (UE4), проте наразі вже наявна версія Unreal Engine 5 (UE5), яка вийшла на момент завершення написання цієї роботи, тому розглядатись не буде. Але, оскільки розробник заявив, що проєкти на UE5 будуть запускатись і на UE4, то можна робити висновок, що ядро рушія не мало суттєвих змін.

3.1.1. Історія розвитку рушія Unreal Engine

Розробкою UE займається компанія Epic Games (EG) [15] (див. рис. 8) яка історично займалась розробкою ігор. Пізніше EG застували свій магазин ігор Epic Games Store [16] який є одним з двох найбільших магазинів комп’ютерних ігор.

Рисунок 8. Логотип Epic Games

Саме під час розробки однієї з ігор був створений перший UE у 1998-му році [17]. Цей рушій видавав дуже якісну, за канонами того часу звісно ж, картинку, оперував великими ігровими рівнями. Проте платою за це стали високі вимоги до потужностей комп'ютерів, що надалі стане «візитівкою» UE.

Невдовзі після цього вийшов UE1.5 у якому покращилась мережева складова, що спростила створення багатокористувацьких ігор.

У 2002-му році вийшов UE2, де переробили деякі механіки всередині рушія і покращили роботу з графікою та фізикою. Продовження у вигляді UE2.5 в який додали підтримку консолей.

Вихід UE3 відзначив не тільки покращення вже наявного функціоналу, але і була випущена безкоштовна версія рушія, яку з радістю почали використовувати різні розробники.

Вже у 2014-му році вийшов UE4 який продовжив розвиток у напрямку комп'ютерної графіки.

5-го квітня 2022 року відкритим для завантаження став UE5.

3.1.2. Мова програмування для розробки

Сам рушій написаний мовою програмування C++ [9]. Дана мова є досить складною для опанування через необхідність розробнику самому керувати пам'яттю, має досить мало вбудованих функцій обчислення що полегшують

розрахунки. Проте це все ще одна з найшвидших мов, тож після опанування програміст зможе писати, за умови відповідного рівня навичок, дуже оптимізований код у тому числі і змінювати логіки базових класів рушія.

Поряд з цим, рушій також підтримує мову скриптів Blueprint. Ця мова підтримує майже весь той самий функціонал, що дає розробнику C++ при цьому завдяки візуалізації у вигляді блоків дає змогу розробнику-початківцю легше зрозуміти, що і як працює.

Серед обмежень для використання Blueprint, як приклад можна навести неможливість змінити базові класи рушія, чи проблеми зі складними обчисленнями, такими як маніпулювання таблицями сотень гравців у багатокористувацьких іграх.

3.1.3. Інструментарій рушія Unreal Engine

Весь наявний у рушії перелік інструментів для створення ігор найвищого рівня доступний розробнику одразу після завантаження рушія [18].

Окрім того, завдяки великій та активній спільноті існує багато різних ассетів, шматочків коду, механік (як на C++, так і на Blueprint) які також доступні розробнику одразу.

Проте не рекомендується використовувати чужий код як головну частину власних механік, оскільки він може бути некоректний у рамках вашої задачі, чи складним у підтримуванні, особливо якщо автор коду продовжує його оновлювати.

Серед прикладів інструментарію наведемо [18]: Landscape – інструмент для створення ландшафту рівня; LOD – інструмент, що дозволяє змінювати деталізацію текстур в залежності від відстані до «камери» на базі пріоритетів.

3.1.4. Ліцензійні умови

Згідно з ліцензійною угодою, ігровий рушій Unreal Engine можна завантажити та використовувати безкоштовно [19]. Проте існує деякий поріг,

після якого використання рушія (а це можуть бути не тільки ігри, а, наприклад, комп’ютерна графіка для фільмів) стає платним.

Коли дохід від продукції у якій використовується UE (це можуть бути не тільки ігри, а, наприклад, комп**’**ютерна графіка у фільмах**)** перевершить 1 000 000 USD, володар продукції має сплачувати 5% від прибутку [19].

3.1.5. Висновки до характеристики ігрового рушія Unreal Engine

Ігровий рушій Unreal Engine розробляє компанія, основною діяльністю якої є розробка ігор. Відповідно, цей рушій спрямований саме на якість кінцевого продукту.

Весь функціонал доступний одразу ж після відкриття рушія, а велика та активна спільнота створює все більше і більше допоміжних матеріалів та активно бере участь у виправленні помилок всередині рушія. Зазначені виправлення не рідко попадають до офіційних оновлень рушія.

Окремо зазначимо великий пільговий період у оплаті за використання рушія, що добре для малих студій. Також позитивним моментом є наявність двох мов для розробки, що полегшує «введення до розробки ігор» початківцям, та велика кількість навчальних матеріалів [20].

Проте, ігровий рушій Unreal Engine має не зовсім інтуїтивно зрозумілий дизайн всередині самого редактора, дуже велику вимогливість у плані ресурсів комп’ютера, що ускладнює розробку на слабких машинах.

3.2. Ігровий рушій Unity

3.2.1 Історія ігрового рушія Unity

Отже, у далекому 2005-му році, рушій Unity [6] від Unity Technology [21] став одним першим рушієм, що був безкоштовним і надавав широкий спектр інструментів [22]. Саме тому Unity наразі має найбільшу спільноту розробників серед усіх рушіїв.

Проте слід розуміти, що Unity Technology [21] займаються розробкою рушія, а не ігор. Відповідно спектр задач перед ними стоїть як зробити багатофункціональний і зрозумілий рушій.

Спочатку рушій Unity був тільки для операційної системи Mac OS, проте згодом почав підтримувати мобільні пристрої, Windows, Linux, та ін.

Все це призвело до величезного стрибка його використанні й на сьогодні він складає значну частину ігор на комп'ютери та більш ніж половина ігор на мобільних пристроях [23].

3.2.2 Мова програмування для розробки

У Unity мовою розробки є C#. Це єдина мова, якою ви можете розробляти ігри на Unity. Проте, будучи ООП-орієнтованою, досить інтуїтивно зрозумілою ця мова не є занадто складною в опануванні.

Окремо хотілось би зазначити, що спільнота створила велику кількість асетів та плагінів, що суттєво полегшують розробку.

3.2.3. Інструментарій рушія Unity

Інструментарій, який пропонує рушій «з коробки» не є вичерпним і не завжди є зручним. Проте у цей момент на допомогу прийшла спільнота користувачів.

Завдяки великому ентузіазму та досвіду роботи з рушієм, учасники спільноти створили дуже велику кількість плагінів, асетів та інших речей. Настільки велику, що в Unity існує власний Asset Store [24].

На жаль не весь функціонал, що є всередині безкоштовний. Проте і безкоштовних речей більш ніж достатньо, щоб суттєво полегшити розробку гри.

Одним з прикладів таких інструментів може бути Dialogue System for Unity [25] – квестовий рушій. Він дозволяє створювати систему квестів та полегшує оперування з діалогами та персонажами.

3.2.4 Ліцензійні умови

Unity пропонує систему підписок [26]. Безкоштовна підписка діє до моменту, поки ваш прибуток не перевищує 100 000 USD на рік.

За умови, що ваш прибуток складає менш як 200 000 USD на рік ви можете придбати підписку Plus, що вартує 399 USD з людини на рік

При прибутках що перевищують 200 000 USD на рік у вас буде вибір, або підписка Pro за 1800 USD з людини на рік, або Enterprise з вартістю 4000 USD за 20 осіб на місяць.

3.2.5 Висновки до характеристики ігрового рушія Unity

Перевагами є кросплатформність, простий в освоєнні C#, велика спільнота, яка невпинно штовхає розвиток рушія вперед.

Платою за це стане можлива неоптимальність процесів рушія у певних місцях та проблеми вибору інструментів серед великого списку.

3.3. Порівняльна характеристика ігрових рушіїв

Для полегшення сприйняття фінальне порівняння виконано у вигляді таблиці (див. табл. 2)

Таблиця 2: Порівняння рушіїв Unreal Engine та Unity

Властивість	**Unreal Engine**	**Unity**
Філософія розробника рушія	Основна мета компанії Epic Game створити інструмент для створення високоякісного кінцевого ігрового продукту, витискаючи максимум можливостей, що може видати комп'ютер гравця і розробника	Основна мета компанії Unity Technologies є створення зрозумілого і функціонального рушія, надаючи розробнику широкий спектр інструментів та навчальних матеріалів
Мова розробки всередині рушія	C++, Blueprint	C#
Інструментарій всередині рушія	Доступний одразу у повному спектрі. Доступні для завантаження інструменти спільноти	Базовий інструментарій не є вичерпним. Наявний широкий спектр, у тому числі безкоштовних, інструментів від спільноти зібраний у Asset Store
Ліцензійні умови	Плата за рушій береться після перевищення порогу заробітку розробника у 1 000 000 USD у розмірі 5% від заробітку	Пропонуються умови місячної\річної підписки починаючи з безкоштовної і до 4000 USD в залежності від заробітку розробника

РОЗДІЛ 4.
ПРИКЛАД РЕАЛІЗАЦІЇ ОСНОВИ ГРИ З ВИКОРИСТАННЯМ ІГРОВОГО РУШІЯ UNREAL ENGINE

У цьому розділі буде покрокова інструкція, де можна знайти основні інструменти та як ними користуватися, щоб зробити основу для простого двовимірного платформера з використанням UE4. Розробка без використання C++, лише на Blueprint.

4.1 Створення анімації

Основною частиною будь-якої гри у сучасному світі є анімація. UE4 маючи картинки з іменами, які включають послідовні номери кадрів у анімації, може створити об’єкт анімації Flipbook у кілька натискань.

Перш за все, завантажуємо нашу картинку до потрібної каталогу всередині проєкту. З точки зору UE4 ваша картинка буде називатись текстурою. Вибравши набір текстур з них можна створити спрайти вибравши необхідний пункт із контекстного меню (див. рис. 9). Зауважу, що спрайти будуть мати такі ж імена, що і породивши їх текстури.

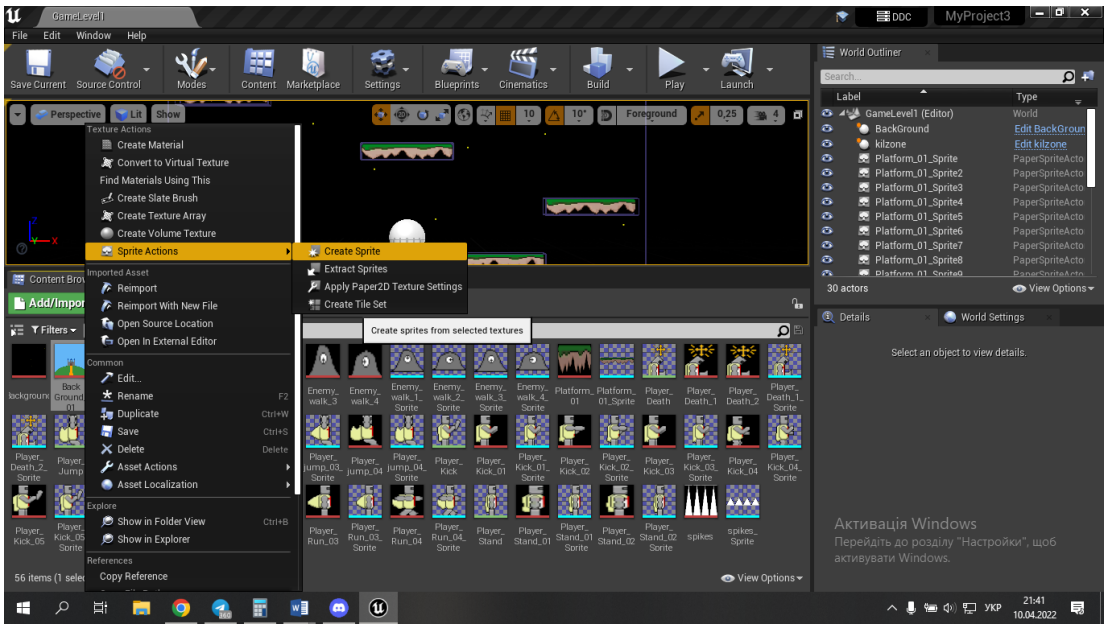


Рисунок 9. Створення спрайту

Вибравши отримані спрайти, всі вони можуть бути сформовані в одну анімацію – Flipbook. За умови правильної нумерації текстур в іменах файлів, а

відповідно і в іменах спрайтів, UE4 одразу ж сформує правильну анімацію. Потрібно лише вибрати правильний пункт у контекстному меню (див. рис. 10)

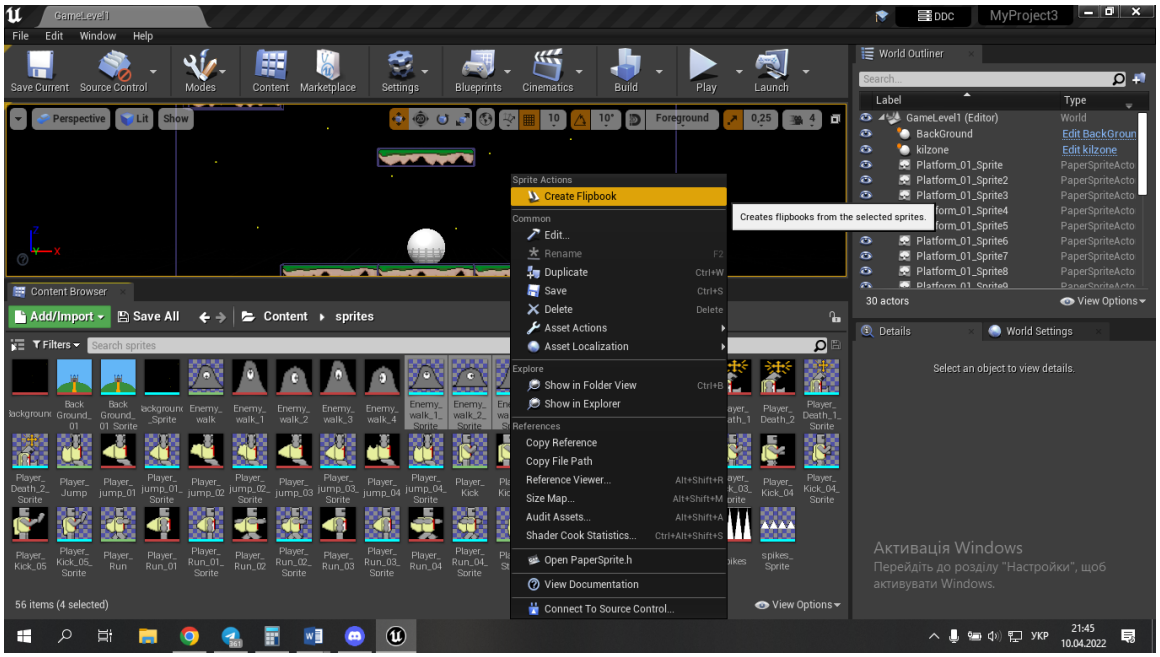


Рисунок 10. Створення Flipbook

Файл Flipbook можна відкрити у редакторі та подивитись правильність порядку спрайтів (знизу зліва) та відредагувати швидкість зміни спрайтів параметром Frames per second (справа зверху) (див. рис. 11)

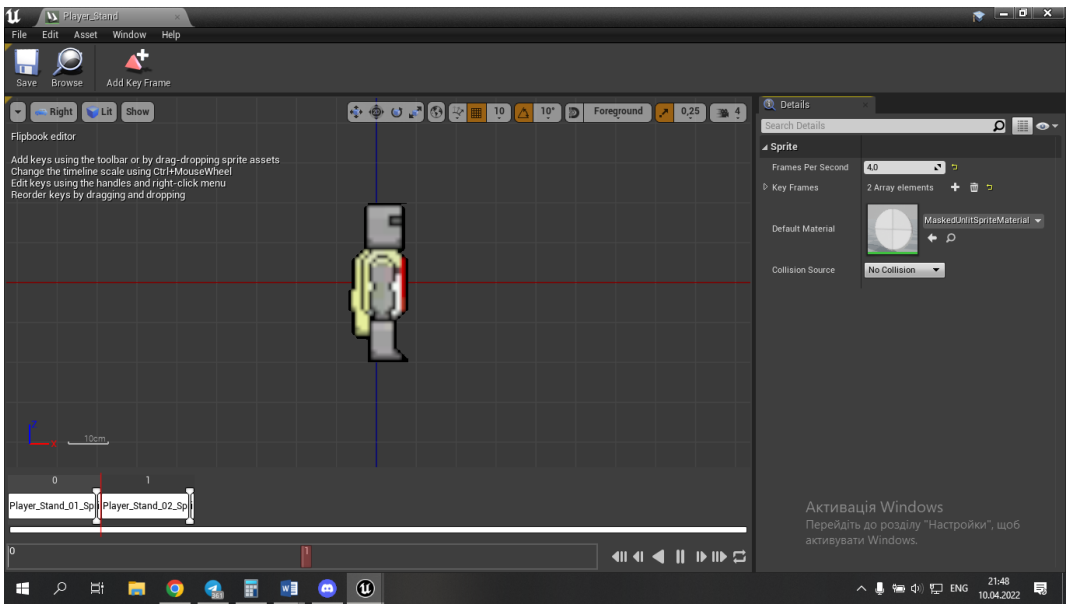


Рисунок 11. Редагування Flipbook

4.2 Створення об’єктів

Оскільки розробка буде виключно з використанням Blueprint то і створювати ми будемо акторів як об’єкти Blueprint Class (див. рис. 12)

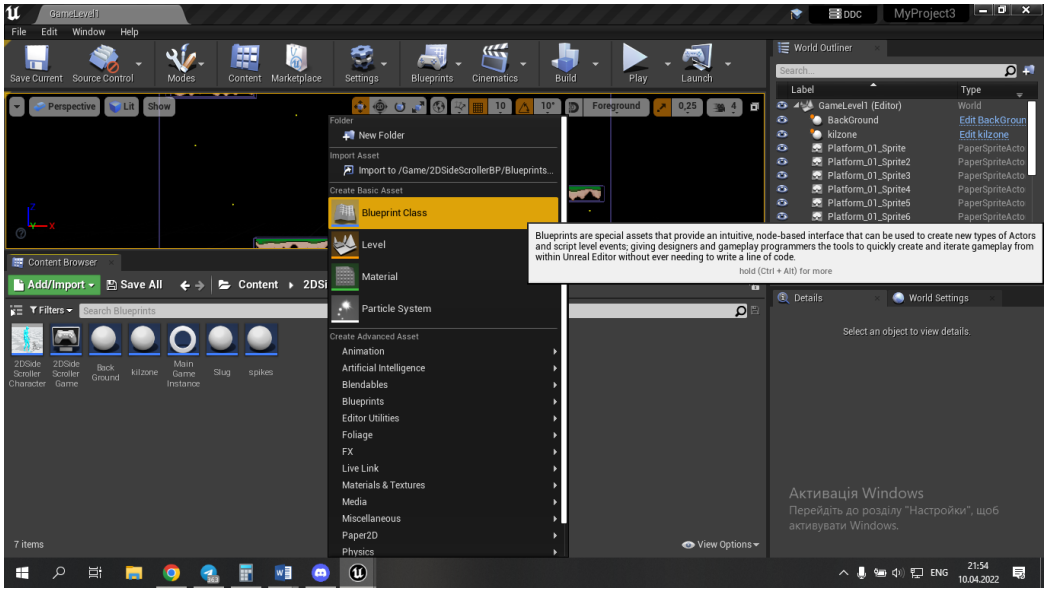


Рисунок 12. Створення Blueprint Class

Створивши об’єкт класу Actor та давши йому ім’я переходимо до редагування(див. рис. 13).

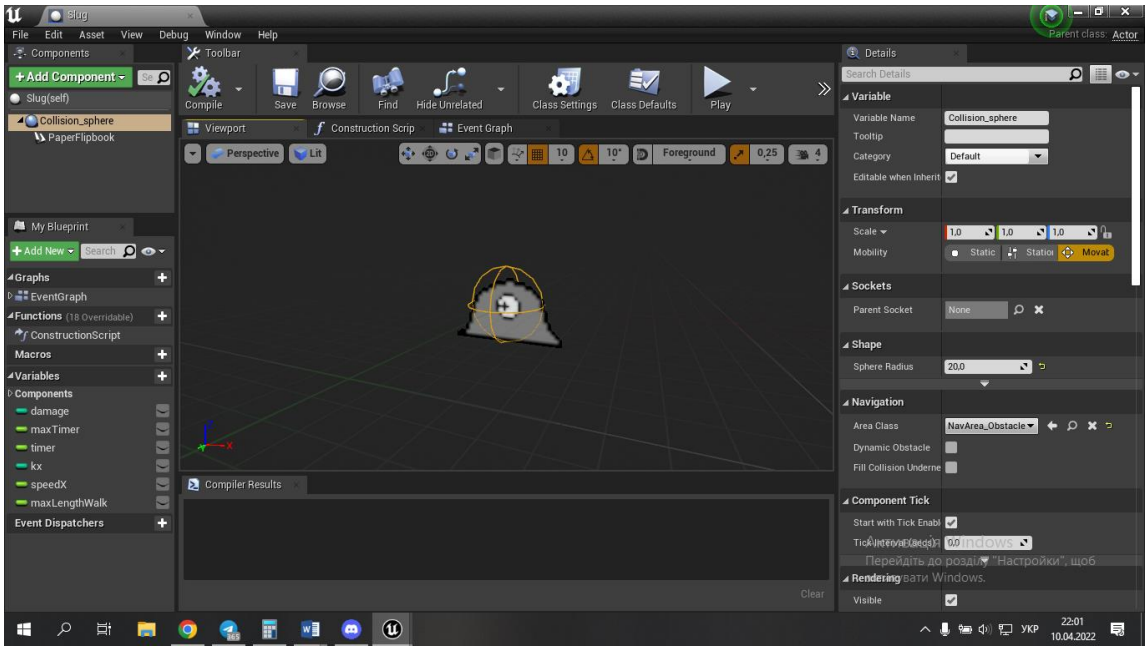


Рисунок 13. Редагування моделі об’єкту

Вибравши на сцені у центрі екрану вкладку Viewport ми потрапимо до редагування фізичної моделі об’єкту. Зліва знизу є місце, до можна додати змінні. Зліва зверху – фізичні об’єкти, як от сфера перетину, чи Flipbook. Вибравши фізичний об’єкт справа ви можете бачити та редагувати його характеристики. Всі ці зміни одразу ж змінять відповідні об’єкти на картинці у центрі екрану.

Якщо перейти на вкладку Event Graph то за допомогою Blueprint можна змінювати логіку дії об’єкта. Основою логіки є Event (подія). Ці події можуть бути базові, як Event Tick, так і створені самим розробником. Під час викликання події рушієм, чи напряму розробником, викликається код, що «підв’язаний» під подію. Всередині такого коду можуть бути викликані інші події.

Приклад реалізації руху можете побачити нижче на рис. 14

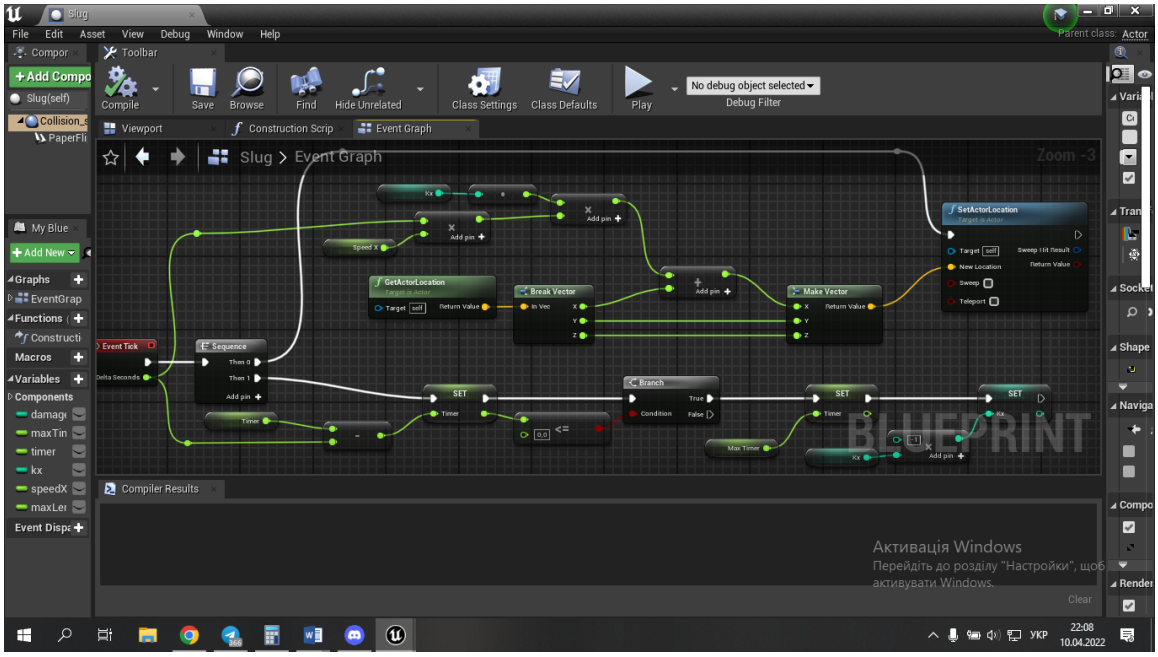


Рисунок 14. Редагування логіки об’єкту

Центральна вкладка Construction Script дає змогу змінювати логіку під час первинної ініціалізації об’єкту (див. рис. **15**)

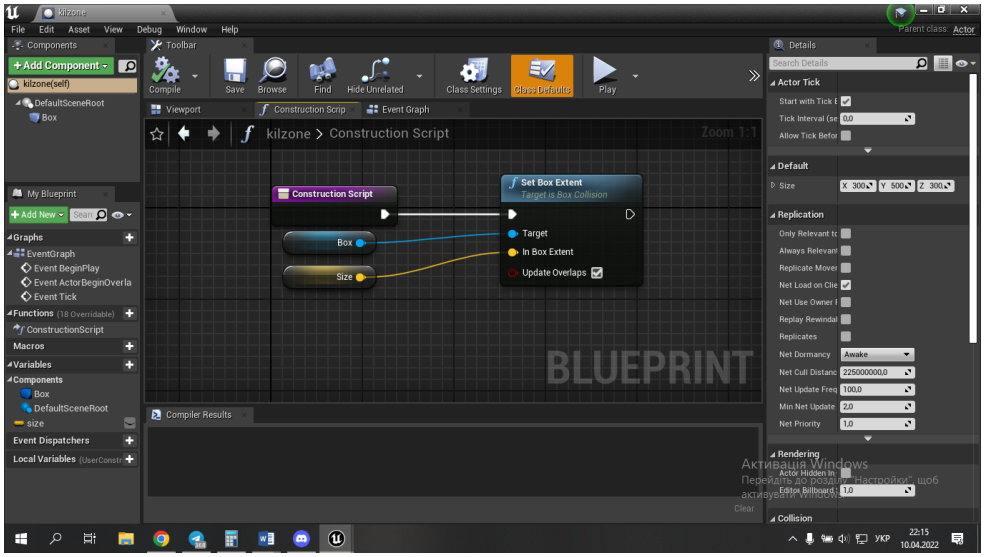


Рисунок 15. Редагування ініціалізації об’єкту

Після редагування об’єктів не забувайте натискати кнопку Compile над вкладками.

Закінчивши з редагуванням Blueprint Class закривайте редактор та додавайте до ігрової сцени необхідні об’єкти, які не задають свої позиції при ініціалізації. Зробити це можна перетягнувши об’єкт Blueprint Class на ігрову сцену та задавши йому координати у правій частині екрану (див. рис. 16)

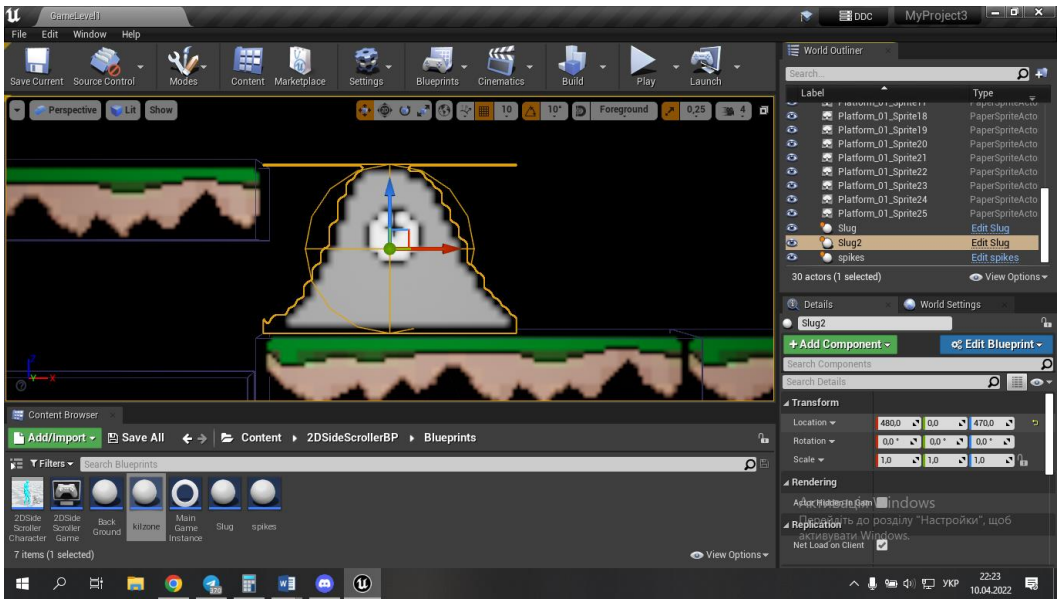


Рисунок 16. Редагування ініціалізації об’єкту

4.3 Приклади реалізації класів з використанням Blueprint

Для прикладу розглянемо реалізацію слизню. Це ворожий герой, який циклічно рухається вправо на 240 одиниць, а потім вліво на 240 одиниць. Під час перетину з гравцем, слизень має вбивати гравця.

Для початку ініціалізуємо змінні, у яких буде збережена потрібна інформація (див. рис. 17).

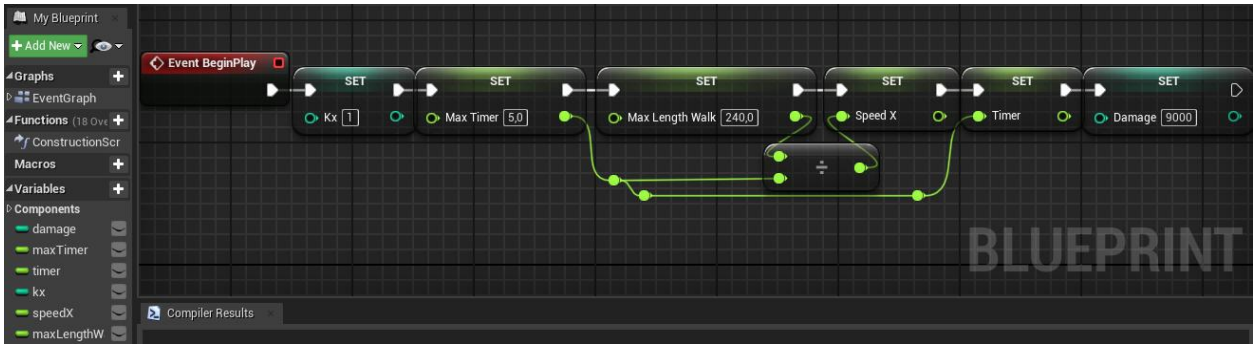


Рисунок 17. Ініціалізація змінних

Тут ініціалізуємо змінні наступними параметрами:

- Kx (напрямок руху) - 1
- MaxTimer (максимальний час руху в одну сторону) - 5 секунд.
- MaxLengthWalk (максимальне переміщення) - 240 одиниць
- SpeedX (швидкість руху в горизонтальній осі) - Максимальне переміщення поділене на максимальний час руху
- Timer (час, який залишилось рухатись до зміни напрямку) - 5 секунд
- Damage (шкода) - 9000 одиниць

Для запису значення використовується блок SET. Біле з'єднання використовується для встановлення черги виконання операторів. Оператори виконуються у порядку "зліва-направо", тобто зліва приєднується попередня операція, справа наступна.

З'єднання знизу використовуються для передачі та повернення параметрів. Ті, що зліва - для передачі, ті, що справа - для повернення.

Передавати параметри можна після розрахунку значень в інших блоків. Прикладом такої передачі є блок Set SpeedX, де значення параметру отримується після розрахунку значення у блоці ділення.

Повертають параметри, наприклад блок Set MaxTimer. Цей блок записує значення у змінну, а потім повертає значення, що було записано. Надалі повернене значення передається як параметр до блоку ділення.

Код починає виконуватись, коли гра передає на об’єкт слизня подію “BeginPlay”, оскільки початком даного ланцюжка коду є червоний блок “Event BeginPlay”.

Взагалі увесь код на Blueprint базується на подіях. Події можуть бути як задані самим рушієм, так і створенні розробником. Аналогічно і викликати події може сам рушій користуючись внутрішньою логікою, так і розробник, додаючи генерацію виклику у свій код.

Наступною частиною буде блок, що відповідає за заподіяння шкоди. Слизень має “вбити” гравця якщо той доторкнеться до моделі слизню.

Для перевірки є Collision Shape. Це деяка геометрична конструкція, яка задає “фізичні” границі об’єкту з точки зору рушія. Серед форм існують сфери, куби та капсули (циліндри з напівсферами на кінцях).

Перевірка на перетин виконується рушієм самостійно, розробнику лише повертається подія ActionBeginOverlap, як і потрібно обробити (див. рис. 18)

Рисунок 18: Обробка перетину

Подія повертає також посилання на інший об’єкт, що ввійшов у перетин за даним.

Для заподіяння шкоди використаємо блок Apply Damage. До цього блоку передаємо параметрами іншого актора, якому і буде заподіяно шкоду, та розмір шкоди.

Оскільки всередині слизня шкода зберігається цілочисельним значенням, а Apply Damage приймає число з плаваючою комою, використаємо блок “.” для конвертації типів.

Останньою частиною буде рух. Нагадаємо, що слизень проходить деякий час вліво, а потім той же самий час вправо, і так до нескінченності.

Розіб’ємо цю задачу на дві. Перша - перемістити об’єкт лизня, друга - перевірити потребу у зміні напрямку.

Розглянемо розв'язання першої задачі (див. мал. 19)

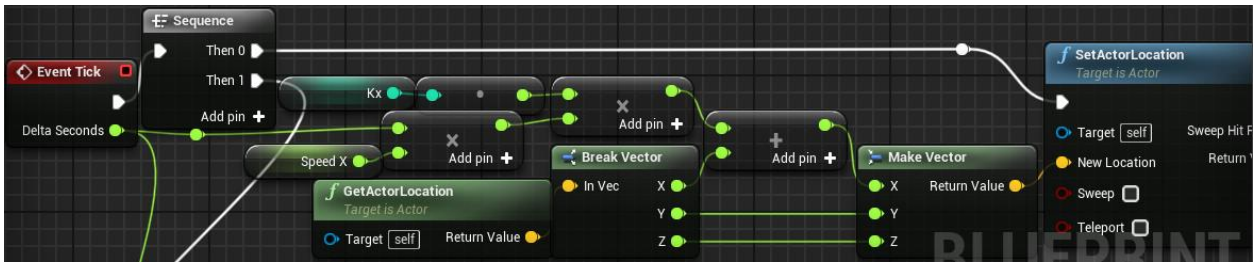


Рисунок 19: Рух слизня

Подія Tick викликається через певні проміжки часу рушієм для виконання рухів та перерахунків станів об’єктів. Окрім того, подія повертає час у секундах, що пройшов з моменту останньої події Tick.

Блок Sequence використання для більш зручного подання розділення на дві під задачі, та не виконує ніяких функцій окрім послідовного виклику двох послідовностей блоків.

Блок Get Actor Location дістає вектор-координат з місцем знаходження об’єкту, для якого виконується даний код. блок Break Vector розбиває вектор на три координати та дає змогу працювати з кожною окремо.

Перемноживши час, швидкість та напрямок руху отримаємо зміну у X координаті. Склавши це значення з наявною координатою отримуємо нове значення.

Передамо ці значення на блок Make Vector що збере всі координати до вектора та передасть його до блоку Set Actor Location. У блоці Set Actor Location базове значення параметру Actor, до якого застосовується зміна є той актор, для якого викликано код.

Роздивимось рішення другої підзадачі (див. рис. 20).

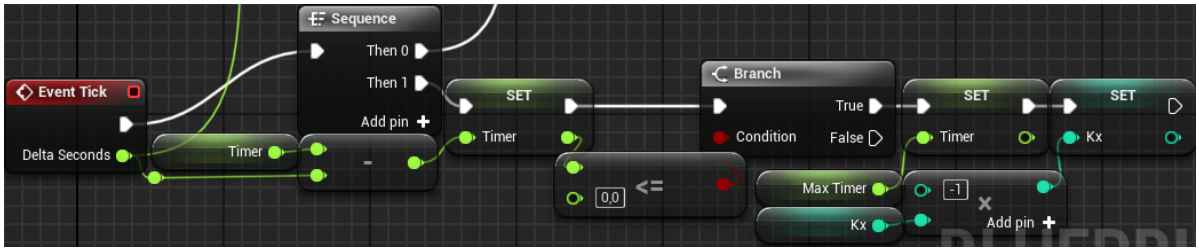


Рисунок 20: Напрямок руху слизня

Для початку потрібно відняти від Timer, що містить час, який залишився до зміни напрямку, час, що пройшов і переписати змінну Timer.

Далі, за умови, що значення змінної Timer менше-рівне нуля, потрібно змінити напрямок руху на протилежний, та оновити значення змінної Timer.

Умовний оператор “If” у Blueprint виконується блоком Branch.

Прописавши всі основні механіки слизню, можна додавати його до гри.

4.4 Висновки з реалізації основ гри за допомогою ігрового рушія Unreal Engine

При використанні рушія Unreal Engine для розробки ігор значно полегшується задачі розробника. Основні об’єкти ігрової логіки можуть бути створені дуже швидко, не потребуючи великих зусиль.

Основною проблемою у такому підході стає створення текстурних зображень. Саме ця частина роботи забирає найбільше часу у процесі розробки гри за допомогою рушія.

Окремо зазначимо, що додавання готових елементів до наявних ігрових сцен також значно полегшено. Полегшення досягаються завдяки простому та зрозумілому інтерфейсу користувача та наочність, оскільки додані об’єкти одразу ж відображаються користувачу на тому місці в ігровій сцені, до них було розміщено.

Завдяки отриманим навичкам у ході виконання роботи є можливість скласти фінальну характеристику і порівняння двох підходів: з використанням ігрових рушіїв, та без них.

Задля полегшення сприйняття порівняння буде подано у вигляді таблиці (див. табл. 3).

Таблиця 3: Порівняння підходів з використанням рушія та без нього

	З використанням рушія	**Без використання рушія**
Переваги	● Пришвидшення розробки продукту шляхом використання розроблених та вбудованих у рушій інструментів. ● Вбудована у рушії підтримка кросплатформності.	● Повний контроль над процесами всередині гри. ● Можливість вибирати оптимальні підходи для розв'язання задач.
Недоліки	● Неможливість змінювати поведінку деяких основних механік рушія. ● Обмеження, що накладає сам рушій.	● Необхідність розробляти всі аспекти гри включаючи керування пам’яттю, тощо. ● Необхідність залучати широкий спектр сторонніх бібліотек та проблеми з підтримкою продукту у разі оновлення використаних бібліотек.

ВИСНОВКИ

Таким чином, у рамках даної роботи було виконано аналіз наявних на ринку ігрових рушіїв та викладені результати аналізу.

Під час написання роботи було виконано наступні завдання:

- Розроблено ігровий додаток без використання ігрових рушіїв.
- Розглянуто та виконано опис основних особливостей ігрових рушіїв Unity та Unreal Engine
- Виконано порівняння основних особливостей ігрових рушіїв Unity та Unreal Engine
- Розроблено основні об’єкти та аспекті ігрової логіки для простої двовимірної гри з використанням ігрового рушія Unreal Engine
- Виконано порівняння двох підходів до створення ігрових додатків - з використанням ігрових рушіїв та без них.

Результати та висновки зробленої роботи викладенні у даній кваліфікаційній роботі.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. World of Tanks encore Engine [Електронний ресурс]: [Вебсайт]. – Режим доступу: https://wotencore.net/en-eu/
2. Wargaming Website for Applicants [Електронний ресурс]: [Вебсайт]. – Режим доступу: https://wargaming.com/en/
3. World of Tanks – Бесплатная онлайн игра про Танки [Електронний ресурс]: [Вебсайт]. – Режим доступу: https://worldoftanks.ru/
4. Windows 2000 - [Електронний ресурс]: [Вебсайт]. – Режим доступу: https://microsoft.fandom.com/ru/wiki/Windows_2000
5. Unreal Engine [Електронний ресурс]: [Вебсайт]. – Режим доступу: https://www.unrealengine.com/en-US/
6. Unity 3D [Електронний ресурс]: [Вебсайт]. – Режим доступу: https://unity.com/
7. Самые популярные бесплатные движки для разработки игр [Електронний ресурс]: [Вебсайт] - Режим доступу https://habr.com/ru/company/timeweb/blog/659891/
8. List of Game engines [Електронний ресурс]: [Вебсайт]. – Режим доступу: https://game-engine.fandom.com/wiki/List_of_game_engines
9. C/C++ и MS Visual C++ 2012 для начинающих / Пахомов Б. І. - БХВ Петербург. - 1 с.
10. SFML [Електронний ресурс]: [Вебсайт]. – Режим доступу: https://www.sfml-dev.org
11. OpenGL [Електронний ресурс]: [Вебсайт]. – Режим доступу: https://www.opengl.org//
12. AI у іграх #1: огляд, настольні ігри (minimax та альтернативи), state machines, behaviour trees [Електронний ресурс]: [Відео]. – Режим доступу: https://www.youtube.com/watch?v=zlEI6ii28_A&t=3849s&ab_channel=ArtemKorotenko

13. Artificial Intelligence for Games / Ian Millington, John Funge.- CRC Press 2020.- 874 с.

14. Репозиторій на GitHub із кодом проекту [Електронний ресурс]: [Веб-сайт]. – Режим доступу: https://github.com/BagInCode/ECO-game

15. Home – Epic Games [Електронний ресурс]: [Вебсайт]. – Режим доступу: https://www.epicgames.com/site/en-US/home?lang=en-US

16. Epic Games Store [Електронний ресурс]: [Вебсайт]. – Режим доступу: https://store.epicgames.com/ru/

17. История Unreal Engine: от первого Unreal до наших дней [Електронний ресурс]: [Вебсайт]. – Режим доступу: https://meownauts.com/unreal-engine-history/

18. Можливості UE4 [Електронний ресурс]: [Вебсайт]. – Режим доступу: https://highload.today/blogs/10-instrumentov-unreal-engine-4/

19. Unreal Engine (UE5) licensing options - Unreal Engine [Електронний ресурс]: [Вебсайт]. – Режим доступу: https://www.unrealengine.com/en-US/license

20. Learn How to Use Unreal Engine - A Powerful Real-Time 3D Creation Platform - Unreal Engine [Електронний ресурс]: [Вебсайт]. – Режим доступу: https://www.unrealengine.com/en-US/learn

21. Wondering what Unity is? Find out who we are, where we've been and where we're going | Unity [Електронний ресурс]: [Вебсайт]. – Режим доступу: https://unity.com/our-company

22. How Unity3D Became a Game-Development Beast [Електронний ресурс]: [Вебсайт]. – Режим доступу: https://insights.dice.com/2013/06/03/how-unity3d-become-a-game-development-beast/

23. Unity — самый популярный игровой движок? [Електронний ресурс]: [Вебсайт]. – Режим доступу: https://dtf.ru/gamedev/861409-unity-samyy-populyarnyy-igrovoy-dvizhok-obzor-dvizhkov-na-kotoryh-delayut-igry-dlya-steam

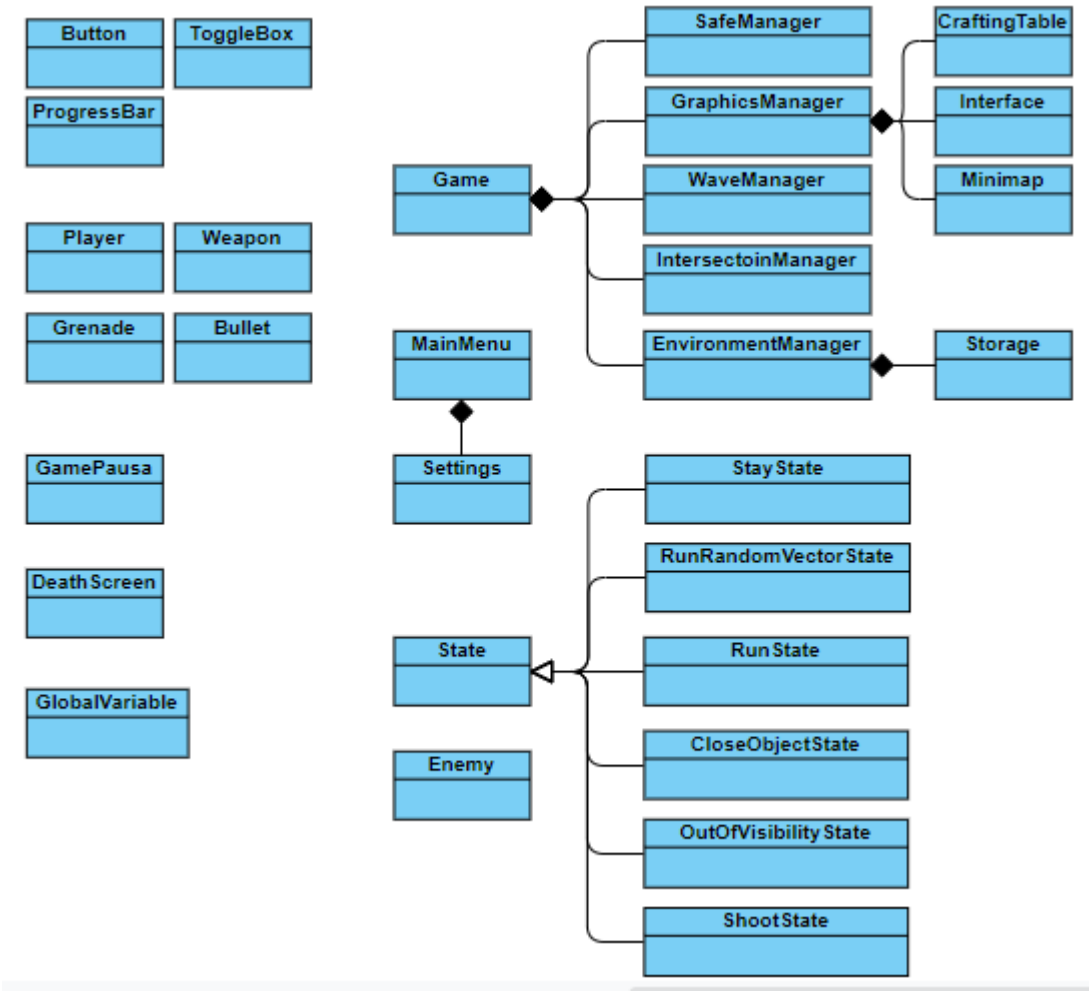
24. Unity Asset Store [Електронний ресурс]: [Вебсайт]. – Режим доступу: https://assetstore.unity.com/

25. Dialog System for Unity [Електронний ресурс]: [Вебсайт]. – Режим доступу: https://assetstore.unity.com/packages/tools/ai/dialogue-system-for-unity-11672

26. Подберите идеальный тарифный план [Електронний ресурс]: [Веб-сайт]. – Режим доступу: https://store.unity.com/ru/compare-plans

ДОДАТКИ

Додаток А. Діаграма класів коду гри



Додаток Б. Приклади «ручної» реалізація ігрового застосунку

Реалізація перевірки на перетин гравця/ворога із будівлями (об’єкт для переходу до режиму покращення зброї, та ангару для збору ресурсів):

```
bool Game::IntersectionManager::orientedArea(double x1, double
y1, double x2, double y2, double x3, double y3)
{
     /*
     * function of getting oriented area of triangle A(x1, y1)
B(x2,y2) C(x3, y3)
     *
     * @param x1, y1, x2, y2, x3, y3 - coordinates of triangle
vertex
     *
     * @return true if oriented area greather than zero
     */

     // calculate area
     double S = x1*y2 + y1*x3 + x2*y3 - y2*x3 - x2*y1 - x1*y3;

     return (S > 0);
}

pair < double, double >
Game::IntersectionManager::calculateNewPosition(double objectX,
double objectY, double objectSize, double xLeft, double xRight,
double yUp, double yDown)
{
     /*
     * function of getting new position to making no
intersection
     *
     * @param objectX, objectY - coordinates of centr of mooving
object
     *        objectSize - size of mooving object
     *        xLeft, xRight, yUp, yDown - coordinates of second
rectangle
     *
     * @return new position of centre of mooving object to
making no intersection
     */

     // get oriented area
     bool flag1 = orientedArea(objectX, objectY, xLeft, yDown,
xRight, yUp);
     bool flag2 = orientedArea(objectX, objectY, xLeft, yUp,
xRight, yDown);

     // if moving up or right
     if (flag1)
     {
          // if up
```

```
            if (flag2)
            {
                return{ objectX, yDown + objectSize / 2. };
            }
            else
            {
                return{ xRight + objectSize / 2., objectY };
            }
        }
        else
        {
            // if left
            if (flag2)
            {
                return{ xLeft - objectSize / 2., objectY };
            }
            else
            {
                return{ objectX, yUp - objectSize / 2. };
            }
        }
}
```

Перетин куль із об’єктами:

```
        // get bullet previous position
        pair < double, double > previousPosition =
bullet.getPreviousPosition();

        // get bullet current position
        pair < double, double > currentPosition =
bullet.getPosition();

        // get delt between positions
        double deltX = (currentPosition.first -
previousPosition.first) / COUNT_SEGMENTS_FOR_BULLET_CHECKING;
        double deltY = (currentPosition.second -
previousPosition.second) / COUNT_SEGMENTS_FOR_BULLET_CHECKING;

        // remember here positions for checking
        vector < pair < double, double > > positionsForChecking;

        // remember it
        for (int i = 0; i < COUNT_SEGMENTS_FOR_BULLET_CHECKING;
i++)
        {

        positionsForChecking.push_back({ previousPosition.first +
(i + 1)*deltX, previousPosition.second + (i + 1)*deltY });
        }
```

Реалізація патерну State Machine для штучного інтелекту ботів:

```
class State
{
protected:
	double timer = 0.000;
	double maxTimer = 0.000;

	int countShoots = 0;
	int maxCountShoots = 0;

	int stateType;

public:
	virtual void doAction(double _timer, Enemy& enemy, Player&
player) = 0;
	virtual int goNext(Enemy& enemy, Player& player) = 0;
	virtual void randomizeState(mt19937* rnd) = 0;

	int getSateType() { return stateType; };
};
```