Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет _____комп'ютерних наук_____
(повна назва)

Кафедра _____Програмної інженерії_____
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

рівень вищої освіти _____перший (бакалаврський)_____

____«Ігровий програмний застосунок у жанрі Roguelite RPG.____
_____Ігрові механіки, 3D графіка, AI»_____
(тема)

Виконав:

студент _4_ курсу, групи _____ПЗПІ-20-2_____

_____Донець Д.С._____
(прізвище, ініціали)

Спеціальність 121 – Інженерія програмного____
_____забезпечення_____
(код і повна назва спеціальності)

Тип програми ____освітньо-професійна_____

Освітня програма ___Програмна інженерія_____

_____
(повна назва освітньої програми)

Керівник __старший викладач Новіков Ю.С.____
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ _____Дудар З.В._____
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет _____комп'ютерних наук_____
Кафедра _____програмної інженерії_____
Рівень вищої освіти _____перший (бакалаврський)_____
Спеціальність _____121 – Інженерія програмного забезпечення_____
(код і повна назва)

Тип програми _____освітньо-професійна_____
Освітня програма _____Програмна інженерія_____
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____
(підпис)

«___»_____2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові _____Донцю Дмитру Сергійовичу_____
(прізвище, ім'я, по батькові)

1. Тема роботи __«Ігровий програмний застосунок у жанрі Roguelite RPG. Ігрові механіки, 3D графіка, AI»_____
Затверджена наказом університету від _20___травня_____ 2024 р. №__471Ст____
2. Термін подання студентом роботи до екзаменаційної комісії 06 _червня_2024 р.
3. Вихідні дані до роботи _____Розробити ігровий програмний застосунок у жанрі Roguelite RPG, а саме ігрові механіки, 3D графіку, та штучний інтелект за допомогою ігрового рушію Unreal Engine 5 та мови програмування C++
4. Перелік питань, що потрібно опрацювати у роботі _вступ, аналізи предметної__ галузі, формулювання вимог до програмної системи, архітектура та проєктування програмного забезпечення, опис прийнятих програмних рішень, тестування розробленого програмного забезпечення, впровадження програмного забезпечення, висновки, перелік джерел посилання, додатки_____

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі	08.04.2024	*виконано*
2	Створення специфікації ПЗ	11.04.2024	*виконано*
3	Проектування ПЗ	15.04.2024	*виконано*
4	Розробка ПЗ	15.05.2024	*виконано*
5	Тестування ПЗ	17.05.2024	*виконано*
6	Оформлення пояснювальної записки	20.05.2024	*виконано*
7	Підготовка презентації та доповіді	21.05.2024	*виконано*
8	Попередній захист	22.05.2024	*виконано*
9	Нормоконтроль, рецензування	25.05.2024	*виконано*
10	Здача роботи у електронний архів	01.06.202	*виконано*
11	Допуск до захисту у зав. кафедри	03.06.2024	*виконано*

Дата видачі завдання 8 квітня 2024 р.

Студент (ка)_____ Донець Д.С.
(підпис)

Керівник роботи _____ старший викладач Новіков Ю.С.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка до кваліфікаційної роботи бакалавра, стор. 63, рис. 16, джерел.12, табл. 3.

КОМП'ЮТЕРНА ГРАФІКА, ROGUELIKE, RPG, ПРОГРАМНИЙ ЗАСТОСУНОК, C++, UNREAL ENGINE

Об'єктом розробки є система тіней [1] і візуальних ефектів, штучного інтелекту та системи заклять, які є частиною комплексної роботи з розробки ігрового застосунку у жанрі Roguelite [2] RPG [3].

Метою розробки є оптимізована система тіней і система тісної взаємодії гравців за допомогою різних заклять у мультиплеєрі, що вписуються у концепт жанра.

Метод рішення – середовище розробки JetBrains Rider, мова програмування C++ та Blueprint, ігровий рушій Unreal Engine 5, середовище 3D моделювання та анімації Blender.

У результаті створено систему тіней, що створюються у реальному часі та систему заклять у мультиплеєрі, що є частиною комплексної роботи з розробки програмного застосунку в жанрі Roguelite RPG.

COMPUTER GRAPHICS, ROGUELIKE, RPG, SOFTWARE APPLICATION, C++, UNREAL ENGINE

The object of development is a system of shadows and visual effects, artificial intelligence, and spell system, which is a part of the complex work of developing a game application in Roguelite RPG genre.

The main goal is to create an optimized shadow system and close interaction system by the means of different spells in multiplayer, which look naturally with given genre.

The means of solution include IDE JetBrains Rider, C++ and Blueprint programming languages, Unreal Engine 5 game engine and Blender 3D modeling and animation tool.

The result is real-time shadow system and multiplayer spell system, which is a part of complex work of developing a game application in Roguelite RPG genre.

Я, Донець Дмитро Сергійович, студент гр. ПЗПІ-20-2, здобувач вищої освіти на першому (бакалаврському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Ігровий програмний застосунок у жанрі Roguelite RPG. Ігрові механіки, 3D графіка, AI», що буде представлена до екзаменаційної комісії для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElAr KhNURE. Усі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови до допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ЗМІСТ

ВСТУП

Відеоігри завжди розвиваються і кожного року з’являються нові унікальні ігри, і одним з найвизначніших жанрів останнього часу став жанр Roguelite, який поєднує у собі елементи стратегії, екшену і у той самий час надає гравцям багато годин контенту завдяки можливості перегравати гру багато разів і не набридати. Багато ігор вирішують це завдяки використанню елементів RPG ігор і системі заклять і прокачок, що не тільки додає грі варіативності а й робить гру доступною для гравців різних рівнів навичок.

Основним завданням є створення системи тіней, а також геймплейної частини, а саме заклять, що повинна працювати в умовах гри на декількох гравців. Системи, що розробляються, мають відповідати новітнім стандартам ігрової індустрії для жанру Roguelite. З огляду на програмну частину вони повинні бути компонентно незалежними, адже система, що розробляється, буде частиною комплексної роботи з розробки ігрового програмного забезпечення в жанрі Roguelite. Розробка гри має показати ефективність використання заклять як частини системи жанру, коли система тіней покаже загальну оптимізацію генерації кадрів.

Областю використання результату роботи буде сфера комп’ютерних ігор та розваг, у якій можна зацікавити як гравців, так і розробників. Ці системи, розроблені в рамках комплексної роботи, мають можливість бути розповсюдженими на широкому спектрі платформ, причому доступ до них може бути наданий безкоштовно або за доступною для гравців ціною, що сприяє подальшому розвитку проєкту.

Створена у результаті комплексної роботи відеогра у жанрі Roguelite RPG матиме декілька локацій та надаватиме можливість гравцю використовувати різні комбінації заклять, що робитиме кожну нову спробу унікальною

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної галузі

Жанр roguelike, який характеризується процедурним створенням підземель, вічною смертю та фокусом на майстерності через повторювані проходження, відродився в останні роки з появою піджанру roguelite. Успадкувавши основні принципи свого батьківського жанру, roguelite додає елементи, які розширюють його привабливість, зберігаючи при цьому основну складність і звикання, які визначають досвід roguelike.

В основі roguelite лежить ретельно розроблений ігровий цикл. Гравці вирушають у пробіжки через процедурно згенеровані підземелля, стикаючись із дедалі складнішими ворогами та босами. Смерть швидка й невблаганна, змушуючи гравців починати заново з мінімальним перенесенням між спробами. Цей цикл смерті та відродження може здатися караючим, але він виховує відчуття майстерності. З кожним пробігом гравці отримують досвід, вивчають схеми ворогів і відкривають потужні комбінації предметів. Знання, отримані з минулих невдач, сприяють майбутнім спробам, сприяючи постійному вдосконаленню та досягненню.

Roguelite відрізняється від свого попередника тим, що додає елементи, які пом’якшують розчарування, яке часто пов’язане з явищем перманентної смерті. Постійні оновлення, навіть поетапні, створюють відчуття прогресу, яке виходить за межі окремих запусків. Ці оновлення можуть приймати форму підвищення характеристик персонажа, нових здібностей або розширеного набору предметів, які зустрічаються під час майбутніх проходжень. Ця постійна система прогресування стимулює повторні спроби, спонукаючи гравців щоразу рухатися далі.

Крім того, roguelite охоплює більшу різноманітність стилів гри. У той час як деякі roguelites зберігають покрокові, стратегічні елементи своїх предків, інші приймають більш орієнтований на дію підхід, що включає бій у реальному часі та швидкий рух. Це поєднання різноманітних стилів гри призначене для ширшої

аудиторії, залучаючи гравців, які, можливо, не насолоджувалися повільним темпом традиційних рогаліків.

Наративний дизайн у жанрі roguelite є ще одним аспектом, який заслуговує на увагу. На відміну від традиційних roguelikes, де історія часто відходить на другий план, roguelites можуть включати багаті оповіді, які органічно розгортаються в кількох проходженнях. Деталі навколишнього середовища, взаємодія персонажів і тонкі підказки, розкидані по всьому процедурно створеному світу, можуть скласти велику історію. Цей фрагментарний підхід до оповіді заохочує кілька разів, винагороджуючи гравців глибшим розумінням історії гри з кожною спробою.

Успіх roguelite полягає в його здатності знайти баланс між викликом і доступністю. Він зберігає ключові принципи жанру roguelike – каральні труднощі, процедурну генерацію та постійну смерть – але додає елементи, які роблять досвід доступнішим і привабливішим для ширшої аудиторії. Постійні системи прогресу, різноманітні стилі гри та потенціал розвитку оповіді сприяють здатності roguelite забезпечувати глибоке задоволення та нескінченно повторюваний досвід.

Одним з яскравих прикладів жанру є гра Dead Cells, одним із найважливіших внесків гри було в його бойовій системі. У ньому ідеально поєднуються елементи дослідження та платформер Metroidvania зі стрімким екшеном roguelite. (рис 1.1.) Під час бою наголошується на плавності та майстерності з акцентом на точних рухах, ланцюжкових комбо та адаптації до шаблонів атак ворога. Це інноваційне злиття жанрів розширило привабливість ігор жанру roguelite, залучивши гравців, яким подобалися аспекти дослідження та прогресу персонажів Metroidvanias, а також виклики та можливість відтворення формату roguelike.

Dead Cells також революціонізували концепцію світового дизайну в рамках roguelite. Хоча процедурна генерація залишається ключовим елементом, взаємопов’язаний світ Dead Cells створюється враження, що він створений вручну. Біоми різні, але бездоганно пов’язані між собою, створюючи відчуття прогресу та відкриттів протягом кожного забігу.

Рисунок 1.1 – Скріншот гри Dead Cells з основними механіками бою

Ярлики, які відкриваються за допомогою постійних оновлень, дозволяють гравцям ефективніше орієнтуватися в світі під час наступних спроб, сприяючи відчуттю панування над навколишнім середовищем. Цей підхід до дизайну світу відрізняється від деяких попередніх рогуелітів, де процедурно згенеровані рівні могли здаватися роз'єднаними та повторюваними. Взаємопов'язаний світ Dead Cells не лише покращує дослідницький аспект, але й підсилює відчуття прогресу гравця.

Мета-прогрес, система, за допомогою якої гравці відкривають постійні оновлення, які зберігаються між заходами, є ще однією сферою, де Dead Cells залишили свій слід. На відміну від деяких roguelites, де постійні оновлення здаються поступовими, Dead Cells пропонує гравцям суттєвий вибір, який суттєво вплине на їхні майбутні забіги. Гравці можуть розблокувати нові навички, мутації, які змінюють ігрову механіку, і креслення потужної зброї. Ця надійна система метапрогресування не тільки стимулює повторні проходження, але й дозволяє гравцям адаптувати свій досвід, спеціалізуючись на певних стилях гри та збірках. Цей рівень волі гравців і налаштувань став наріжним каменем багатьох сучасних

roguelites, безпосередньо натхненних системами, вперше впровадженими в Dead Cells.

1.2 Аналіз конкурентів

Для визначення задач необхідно спершу переглянути конкурентів та з’ясувати переваги та недоліки кожного з них.

Однією з ігор для розгляду є відома Enter the Gungeon [4]. Гра відтворює неймовірне підземелля: ви досліджуєте лабіринт підземного світу, наповнений ворогами та кулями, маючи в руках арсенал божевільної зброї, що постійно зростає. (див. рис. 1.2)

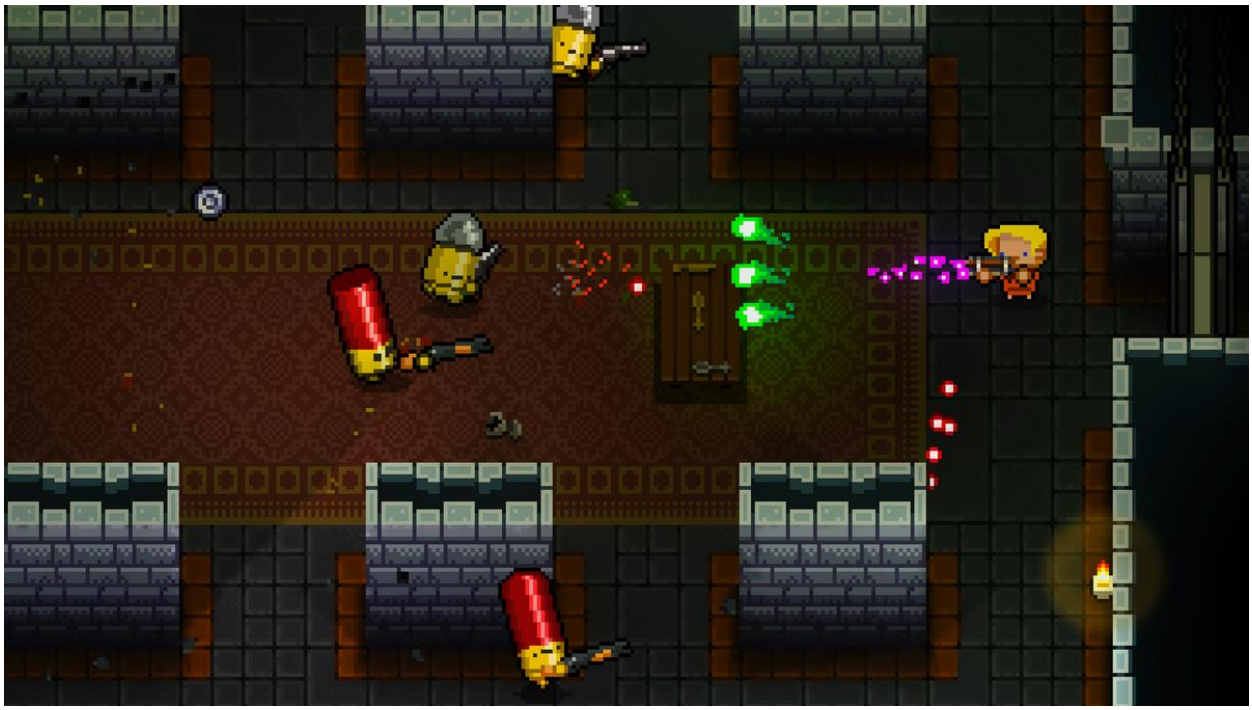


Рисунок 1.2 – Скріншот з гри Enter the Gungeon (2016)

Серед переваг можна зазначити:

- Стрімкий і несамовитий екшн, адже Enter the Gungeon закине вас прямо в саму гущу подій із напруженими перестрілками. Щоб вижити, вам знадобляться швидкі рефлекси та різкі навички ухилення.

- Купа зброї, бо гра може похвалитися величезним арсеналом дивних і чудових зброї, від класичних пістолетів до химерних створінь, які стріляють кулями, що відскакують, або вибуховими вівцями.

- Відтворюваність завдяки можливості гри у кожному проходженні містити процедурно згенеровані рівні та предмети, що зберігає досвід свіжим і складним.

- Спільна гра, тобто можливість кооперативної гри у грі.

Недоліки:

- Висока складність для того, щоб увійти в гру, бо, як відомо, важко, з невблаганними ворогами та системою постійної смерті, яка змушує вас починати з нуля після поразки.

- Повторюваний ігровий цикл, адже для деяких гравців основний ігровий цикл, пов’язаний з битвою в кімнатах і збиранням здобичі, може повторюватися.

Більш новим прикладом успіху була гра Hades [5]. Ця гра, розроблена Supergiant Games, є рольовою екшн-грою 2020 року, яка поєднує в собі несамовиті битви із насиченою розповіддю, орієнтованою на персонажів. Гравці беруть на себе роль Загрея, сина грецького бога Аїда, який намагається втекти з підземного світу та досягти гори Олімп. (див. рис. 1.3)

Рисунок 1.3 – Скріншот з гри Hades (2020)

Переваги:

- Переконливий наратив, який гра майстерно демонструє через взаємодію персонажів і деталі навколишнього середовища. Структура roguelike дозволяє історії органічно розгортатися в кількох проходженнях, винагороджуючи наполегливість.

- Чудова графіка та аудіо, виражена у грі через представлений яскравий художній стиль, натхненний грецькою керамікою, і захоплюючий саундтрек, який динамічно адаптується до ігрового процесу.

- Висока можливість перегравань завдяки поєднанню процедурно згенерованих рівнів, постійних оновлень і розгалуженої розповіді заохочує гравців повертатися за новими.

Мінуси:

- Повторюваний цикл основної гри, хоча історія розгортається протягом кількох ігор, основний цикл гри, повзання в підземеллях і бій, може здаватися деяким гравцям повторюваним.

- Висока складність особливо для тих, хто не знайомий із жанром roguelike, Hades пропонує серйозний виклик, який може збентежити деяких.

Іншим більш класичним прикладом є The Binding of Isaac [6] (див. рис. 1.4), створений Едмундом Макмілленом і Флоріаном Хімслом у 2011 році, — це гра підземелля, із тривожно похмурим естетичним і тематичним ядром.

Рисунок 1.4 – Скріншот гри The Binding of Isaac (2011)

Гравці керують Ісааком, молодим хлопцем, який спускається в підвал, щоб уникнути своєї фанатично релігійної матері, яка вірить, що Бог вимагає жертви. Гра досліджує зрілі теми ізоляції, жорстокого поводження та психічних захворювань через гротескно чарівні візуальні ефекти та ігровий процес.

Переваги:

- Глибокий і символічний наратив, який незважаючи на тривожні образи, The Binding of Isaac пропонує напрочуд глибоку історію про травму та стійкість. Гравці можуть інтерпретувати символіку в грі, щоб розкрити шари значення.

- Поєднання процедурної генерації та різноманітності предметів забезпечує майже нескінченну кількість ігрових вражень.

- Унікальний візуальний стиль, намальований від руки, хоч і викликає занепокоєння, але, безсумнівно, виділяється та створює незабутню атмосферу.

Недоліки:

- Тривожний вміст у вигляді зв’язування графічних образів Ісаака та дослідження темних тем може відштовхнути деяких гравців.

- Загадкова розповідь, яка значною мірою покладається на символізм і вимагає значних інвестицій гравця, щоб її повністю зрозуміти.

- Невблаганний характер жанру roguelike у поєднанні з відсутністю підказок у грі може дуже ускладнити процес навчання.

Таким чином можна прослідкувати проблему складності гри на початку, яку потрібно згладити тимчасовою легкістю і частіше нагороджувати гравця різними способами протягом гри. У цьому може допомогти гарно продуманий сюжет, що легко вплітається у естетику світу та різноманіття секретів, що гравець може знайти по ходу гри.

1.3Постановка задачі

У ході розробки ігрового застосунку у жанрі Roguelite RPG, а саме його геймплейної та графічної частин, у які входять системи тіней та заклять, повинні бути розроблені наступні компоненти:

Компонент заклять:

- потрібно розробити закляття, які відповідають ігровому оточенню та являють собою частину ігрового світу;
- закляття повинні бути різноманітними і впливати на різні атрибути гравця;
- закляття повинні мати декілька рівнів сили і залежати від рівня / досвіду гравця;
- закляття мають працювати у режимі мультиплеєру без значної для геймплею затримки;

Система тіней:

- шейдери повинні працювати у обраному ігровому рушії;
- повинні виглядати не гірше за тіні обраної якості у ігрового рушія;

Штучний інтелект:

- повинен контролюватися сервером;
- повинен отримувати інформацію про гравця;
- кожен ворог має унікальні закляття;
- схожі вороги діють схоже;

Таким чином маємо три основні системи, що вимагають подальшої реалізації в умовах комплексної роботи.

2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

2.1 Функціональні вимоги до ігрового застосунку

З розгляду до постановки задачі у до цього та, з врахуванням переваг та недоліків розглянутих аналогів, для компоненту ігрового застосунку, шо розроблюватиметься, ставляться наступні функціональні вимоги:

Система заклять:

- закляття мають залежати від атрибутів і ставати сильнішими залежно від атрибутів;

- лише закляття можуть впливати на атрибути і таким чином на персонажів гри, тобто сповільнення має впливати на атрибут швидкості руху, шкода на атрибут здоров’я і так далі.

- усі доступні закляття мають відображатися у інтерфейсі гравця.

- закляття мають час перезарядки та вартість, де вартість може бути виражена певною кількістю конкретного атрибуту.

- атрибути впливають на швидкість перезарядки, вартість й інші аспекти закляття, що закладені у його.

Система тіней:

- тіні і пов’язані з ними ефекти не мають значно сповільнювати роботу гри

- тіні і пов’язані з ними ефекти можна легко замінити під час гри

- тіні виглядають достатньо реалістично для гравця

Штучний інтелект:

- вороги повинні завжди переслідувати гравця, або займати пріоритетні позиції з яких легко атакувати гравця.

- вороги повинні використовувати найсильніші закляття у пріоритеті. Такі закляття можуть визначатися умовами у яких знаходиться ворог та гравець відповідно.

Таким чином сформовані функціональні вимоги до частини комплексної кваліфікаційної роботи.

2.2 Нефункціональні вимоги до ігрового застосунку

Ігровий застосунок повинен працювати з мінімально можливими затримками навіть за умови знаходження на одній ігровій сесії декількох гравців одночасно. Також, продуктивність не має залежати від кількості одночасно використаних заклять, чи кількості ворогів, за умови що кількість ворогів не перебільшує 100 на одну ігрову сцену.

Також, до нефункціональних вимог частини ігрового застосунку Roguelite RPG, а саме до компоненту системи заклять, тіней та штучного інтелекту можна віднести:

- виділення ворогів червоним кольором при наведені на них
- спеціальні кольори виділення для різних типів предметів;
- об’єкти та закляття що належать до одного атрибуту мають мати колір цього атрибуту;
- усі об’єкти повинні бути реалістичних пропорцій, не враховуючи ворогів та гравця.

Таким чином сформовані нефункціональні вимоги до частини комплексної кваліфікаційної роботи.

2.3 Вимоги до середовищ розробки

Для створення гри має використовуватися ігровий рушій Unreal Engine 5, як найбільш просунутий рушій з широким інструментарієм, який полегшує задачу створення мультиплеєрних ігор, у той самий час надаючи змогу модифікувати шейдери і те як працюють тіні.

Unreal Engine 5 використовує мову програмування C++ як стабільну та швидку для гри, у цей час підтримуючи інтеграцію JetBrains Rider для легкого

розширення наявного функціоналу, так і для зміни частин вихідного коду ігрового рушія.

3 АРХІТЕКТУРА ТА ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 UML проєктування програмного забезпечення

Розробка комплексної роботи ігрового застосунку розпочалась з UML-проектування. На першому етапі була створена Use Case діаграма, яка візуалізує функціонал гри та демонструє всі можливі взаємодії елементів. Загальна Use Case діаграма для ігрового застосунку у жанрі Roguelite RPG представлена на рисунку 3.1.

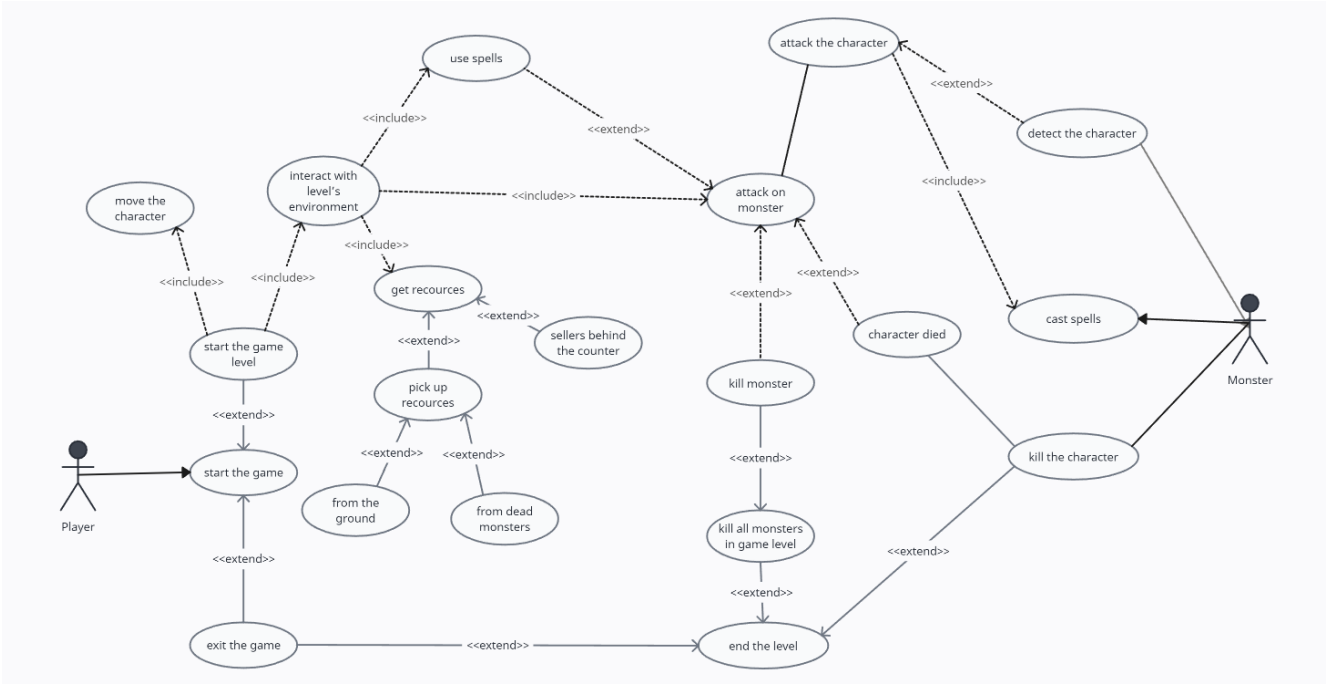


Рисунок 3.1 – Use Case діаграма ігрового застосунку

Розглянемо деякі елементи діаграми, що відносяться до системи заклять та штучного інтелекту більш детально.

Серед акторів розроблюваного компоненту виділимо Монстра (Monster) та Гравця (Player).

Гравець (Player) є безпосереднім користувачем системи та представлений у грі персонажем головного героя.

Монстр (Monster) – це один з видів ворогів у грі, що не є ігровим персонажем. Він доповнює ігровий світ та використовується у деяких механіках:

- знайти персонажа (detect the character) – штучний інтелект знаходить гравця за умови, що той знаходиться у радіусі знаходження та його можна побачити з точки у якій знаходиться монстр.

- атакувати персонажа (attack the character) – штучний інтелект атакує гравця використовуючи закляття (cast spells), які впливають на його атрибути, або на атрибути самого монстра. Таким чином можна, наприклад, наносити шкоду ворогу.

- вбити персонажа (kill the character) – штучний інтелект може вбити гравця, після чого він знаходить нову ціль та атакує її.

Тепер розглянемо Гравця (Player):

- почати гру (start the game) – гравець може розпочати гру і таким чином почати взаємодію з ігровим світом і виконувати будь-які дії в ньому.

- почати рівень гри (start the game level) – гравець починає битву з ворогами і отримує необхідні

- взаємодіяти з оточенням рівня (interact with level’s environment) – гравець може підбирати предмети та бонуси, що дозволяють відкривати нові закляття для використання у подальшому

Отже, у цьому підрозділі були розглянуті варіанти взаємодії для Монстру та для Гравця з боку використання заклять.

Гравець і Монстр можуть взаємодіяти один з одним використовуючи закляття для зміни атрибутів, при цьому основною метою обох є зменшення здоров’я для перемоги над опонентом.

3.2 Вибір архітектури та рушія

На сьогодні існує багато ігрових рушіїв, за допомогою яких можна створювати якісні ігри. Але найбільш популярними рішеннями наразі є три рушія:

Unreal Engine, Unity та Godot. Кожен з них має свої переваги й недоліки, і там де одному проєкту буде більш ефективно використовувати один рушій, іншому набагато легше досягти своєї задачі з іншим.

Unreal Engine на відміну від двох інших кандидатів надає набагато більше функціоналу пов'язаного з графікою та анімаціями, полегшує роботу створення оточення надає можливість змінювати архітектуру рушія на низькому рівні. Програмування на C++ дозволяє досягти додаткового рівня оптимізації. Теж саме стосується й шейдерів, бо Unreal Engine надає можливість редагувати шейдери за допомогою мови HLSL, що застосовується для програмування шейдерів на низькому рівні. У цей час як Godot, так і Unity використовують для програмування геймплею виключно мову C#, або іншу власну скриптову мову, на чому втрачаємо оптимізацію, яка необхідна у іграх.

Однією з найважливіших переваг Unreal Engine 5 є його передова технологія візуалізації. Такі функції, як Lumen [7], повністю динамічна система освітлення, і Nanite [8], віртуалізатор геометрії, дозволяють створювати неймовірно деталізовані та захоплюючі середовища. Це особливо важливо для жанру Roguelite RPG, де гравці занурюються у фантастичні світи, наповнені різноманітним середовищем. Здатність Unreal Engine 5 відтворювати ці світи з такою точністю може значно покращити досвід гравця, створюючи відчуття дива та відкриття в кожному процедурно згенерованому підземеллі.

Крім візуальних елементів, Unreal Engine 5 чудово створює динамічні та реактивні світи. Його система візуальних ефектів Niagara дає змогу створювати вражаючі анімації заклинань, екологічні небезпеки та вражаючі здібності персонажів. Це ідеально поєднується зі стрімкими, орієнтованими на дії боями, які часто зустрічаються в рольових іграх Roguelite. Unreal Engine 5 дозволяє розробникам створювати візуально вражаючі та вражаючі бойові зіткнення, залучаючи гравців і занурюючи їх у дію.

Тепер розглянемо види архітектури і визначимо найбільш придатну з них для створюваного проєкту. Є три основні патерни за якими створюється архітектура ігрових застосунків:

- MVC, така що складається з частин Model, View та Controller. Ці компоненти відповідають за збереження даних, відображення даних та обробку дій користувача. І хоча частково використати цю модель для частин програми можна, вона не є основною. Цю архітектуру буде застосовано для рендерінгу UI і основної інформації, що пов’язана з персонажем і його закляттями.

- Клієнт-серверна архітектура, одна зі стандартних абстрактних архітектур, де одна частина відображається на пристрої користувача, коли інша виконується на віддаленому пристрої, на сервері. Така архітектура використовується для онлайн ігор, але більшу частину онлайн зв’язку і мультиплеєр можливостей, що потребують розділення на клієнт та сервер за цією архітектурою вже застосовано у ігровому рушії, що було обрано. Саме тому дана архітектура не буде застосована.

- Компонентна архітектура, яка розбиває логічні частини на компоненти. Таким чином можна розбити закляття, персонажів та штучний інтелект на компоненти, які можна повторно застосовувати у інших компонентах і ефективно тестувати кожен окремо. Це досить полегшує процес розробки і саме тому ця архітектура буде застосована в процесі розробки.

Завдяки обраній архітектурі легко створювати новий функціонал і не заважати програмуванню іншим учасникам команди. Одночасно з цим застосовується MVC для частини, що відповідає за відображення інтерфейсу, але вона не складає основного функціоналу програми.

3.3 Огляд ігрового циклу

Один з ключових етапів розробки ігрових застосунків – це проектування ігрового циклу (англ. "game loop"). Ігровий цикл описує основні механіки гри та досвід, який гравець отримує під час геймплею . Цей цикл будується за принципом, де кожна дія призводить до наступної, створюючи послідовність, яка в результаті повертається до початкової дії. Таким чином формується геймплей будь-якої гри.

Крім того, деякі дії можуть забирати або додавати ресурси гравцеві, такі як час або ігрова валюта. Розглянемо ігровий цикл розроблюваного ігрового застосунку (див. рис. 3.2). Оскільки ігровий цикл є замкнутим, то позначимо, що гравець починає свою ігрову сесію з події "початок нового рівня" (start new level). Потім відбувається підцикл, відомий як проходження рівня (level walkthrough).

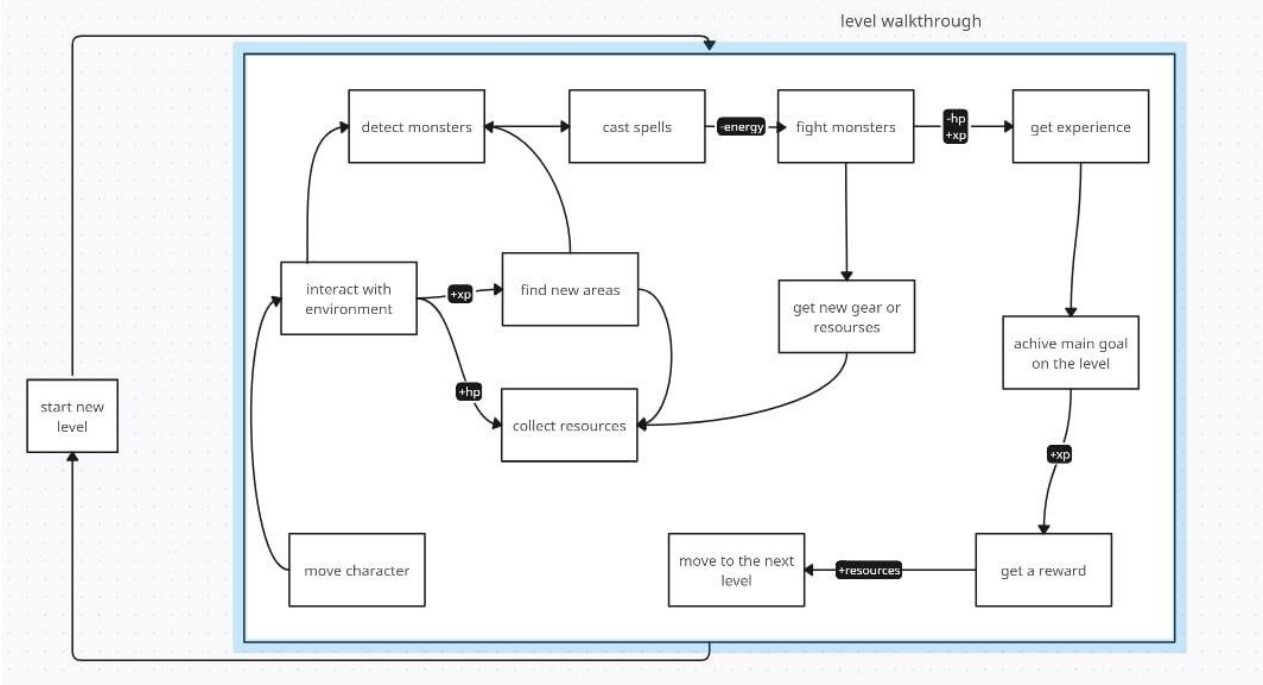


Рисунок 3.2 – Ігровий цикл програмного продукту

Більшість геймплею можна побачити всередині циклі, основні дії, що відбуваються протягом наведеного циклу:

- рух персонажа;
- взаємодія з оточенням;
- відкриття нових територій;
- здобуття нових предметів;
- знаходження ворогів;
- використання заклять;
- отримання досвіду;
- отримання нагороди;

При досягненні основної мети рівня, гравець отримує нагороду і переходить до наступного рівня, де повторює наведені дії. Для досягнення мети гравець має навчитися битися з монстрами та посилити свої здібності, для перемоги над певними ворогами.

3.4 Огляд найцікавіших алгоритмів та методів

В якості цікавого алгоритму буде наведено алгоритм поведінки одного з ворогів [9]. Цей алгоритм є більш комплексним і складним, але його розгляд потребував би більшого обсягу, тому буде розглянуто його частину, яку можна побачити на рисунку 4.3.

Спершу ворог перевіряє чи знаходиться гравець у радіусі знаходження ворогом. Цей процес повторюється поки гравець не з’явиться у радіусі, адже умови рівня змушують гравця знищити ворогів якнайшвидше. Після цього ворог аналізує своє оточення і чи може він рухатися над певними об’єктами. Ворог створює оцінку для кожної точки у певному радіусі з урахуванням шансу того, що він зможе застосувати закляття з цієї точки проти гравця. Після цього аналізу інформація про нього зберігається і ворог починає рухатися у сторону гравця. Якщо ворог досяг необхідної точки, він атакує гравця, застосовуючи закляття проти нього. Цей процес повторюється доки гравець живий. Як тільки гравець помирає, ворог перестає існувати.

Запропонований алгоритм вирішує проблеми, коли ворог може перестати рухатись, тобто застрягти на місці і намагатися продовжити попередню дію, коли її вже неможливо зробити, а також завжди знає про те, чи є гравець у радіусі власного знаходження, чим полегшує задачу атаки гравця, або використання інших заклять відповідно до ситуації, сили заклять та інших впливових факторів, що мають вагу у прийнятті рішень. Таким чином ми можемо гарантувати, що будь-який ворог на сцені буде виконувати призначену йому роль, чи то допомагати

іншим монстрам досягти гравця, чи то використати його оточення більш ефективно й атакувати гравця заздалегідь, чи атакувати його з місця, де вже знаходиться ворог.

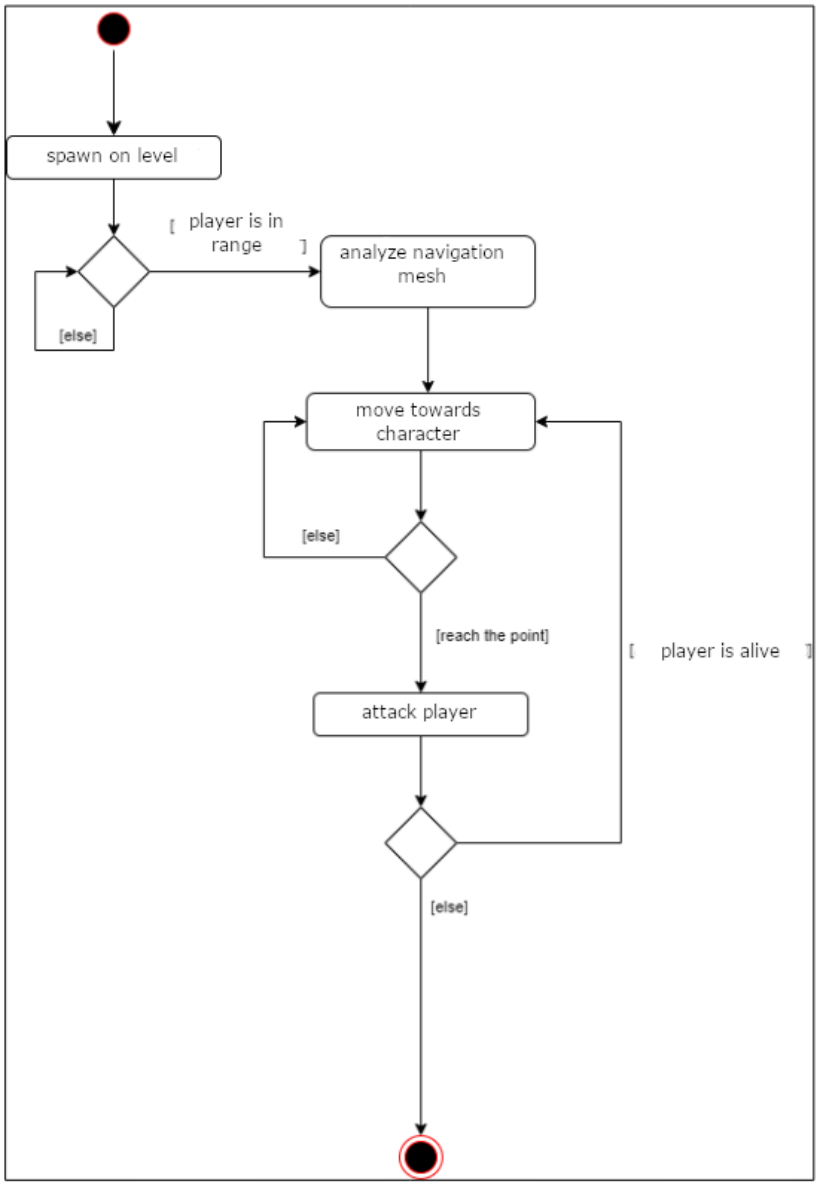


Рисунок 3.3 – Action-діаграма алгоритму поведінки ворога

Наведений алгоритм у діаграмі (рис. 3.3) демонструє вищеописані дії у більш узагальненому форматі, де фактично демонструється можливість будь-яким ворогом досягти гравця, що є однією з найбільш пріоритетних задач, що має виконувати кожен ворог, незалежно від класу і набору здібностей, або інших важливих атрибутів.

3.5 Створення прототипів UI/UX

Для покращення досвіду гри і більш свідомого занурення у світ гри, було використано декілька рішень для покращення інтерфейсу, враховуючи існуючі аналоги.

Гравцю необхідна якнайбільш повна інформація [10] про стан гри та ігрове оточення, тому потрібно передавати майже усю інформацію про активні ефекти та наявні закляття у доступі гравця. (рис. 3.4)

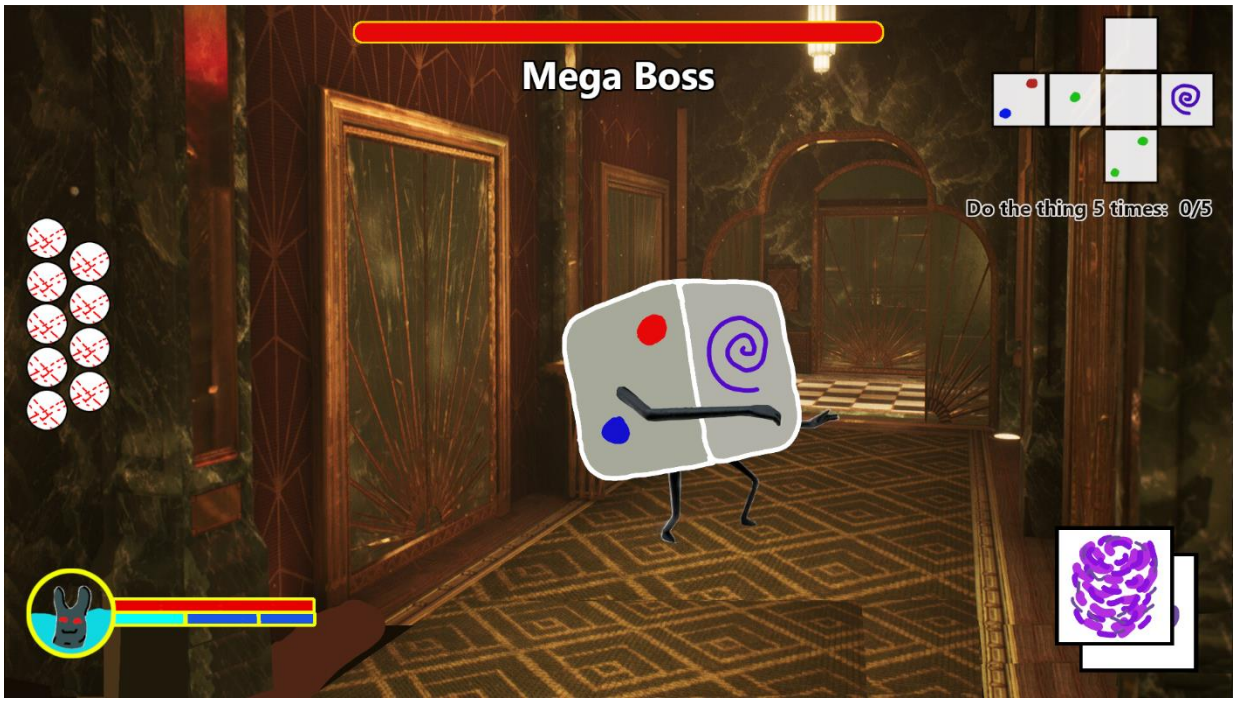


Рисунок 3.4 – Прототип UI під час проходження рівня

На рисунку 3.5 можна побачити декілька основних атрибутів, серед яких три основні елементи – шкалу здоров’я (червону стрічку), шкалу мани (синю стрічку) та шкалу героїчної здібності (синю сферу). Кожен з атрибутів показує основну інформацію про те, як варто гравцю продовжувати гру, чи відштовхуватися від героїчної здібності, чи зіграти більш акуратно і заощадити здоров’я, чи застосувати ще одне закляття, бо мана нещодавно поповнилася.

З лівого боку рисунку 3.4 можна побачити активні ефекти, що або шкодять герою, або допомагають йому, а також скільки залишилося очікувати кінець ефекту, або переконатися у тому що ефект вічний.

Рисунок 3.5 Шкала основних життєвих атрибутів гравця

Правий нижній куток рисунку 3.4 показує активні закляття та можливість їх застосування на даний момент. Це може допомогти прийняти необхідні стратегічні рішення, щоб перемогти ворога на важливому етапі, так як закляття застосовуються набагато частіше аніж героїчна здібність.

У результаті маємо прототип, що містить усю необхідну для гравця інформацію для прийняття важливих ігрових рішень з приводу перемоги над ворогами і просуненню у сюжеті та загальному прогресі.

4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

4.1 Система заклять та здібностей

Основною задачею цієї частини комплексної роботи зазначеного ігрового додатку було створити декілька продуманих алгоритмів, а також архітектури в цілому за допомогою ігрового рушія UE5. Це включало в себе створення системи заклять, що легко розширюється, системи генерації високоякісних тіней та штучного інтелекту, що дає достатньо гарний опір гравцеві у реальному часі, не створюючи занадто складного опонента.

Для початку буде розглянуто архітектуру системи заклять та здібностей, що легко масштабується. Дана архітектура включає в себе багато компонентів, основними з яких є компоненти компонент здібностей (“AbilitySystemComponent”), геймплейного ефекту (“GameplayEffect”), множини атрибутів (“AttributeSet”) та геймплейних міток (“GameplayTags”). Кожен актор у сцені, що має взаємодіяти у геймплейному циклі містить компонент здібностей, що надає йому множину атрибутів та дозволяє застосовувати та отримувати геймплейні ефекти. Таким чином кожен актор може отримувати інформацію про інших акторів за необхідності та взаємодіяти з ними, змінюючи атрибути, які напряму вказують на статус персонажа чи гравця, тобто вказують чи живий він, чи може застосовувати закляття для застосування ефектів, чи здатен переміщуватись чи робити будь-які інші речі, зазначені межами геймплейного циклу. Приклади закляття ворогів можна побачити на рисунку 4.1. Для спрощення взаємодії були застосовані геймплейні мітки, що найчастіше вказують на наявність певного ефекту у персонажа, чи він у стані приголомшення, засліплення, чи зловив вдачу. Завдяки ним ми можемо перевірити невразливість до певних ефектів, чи навпаки можливість накладати чи отримувати певні ефекти від наданих заклять.

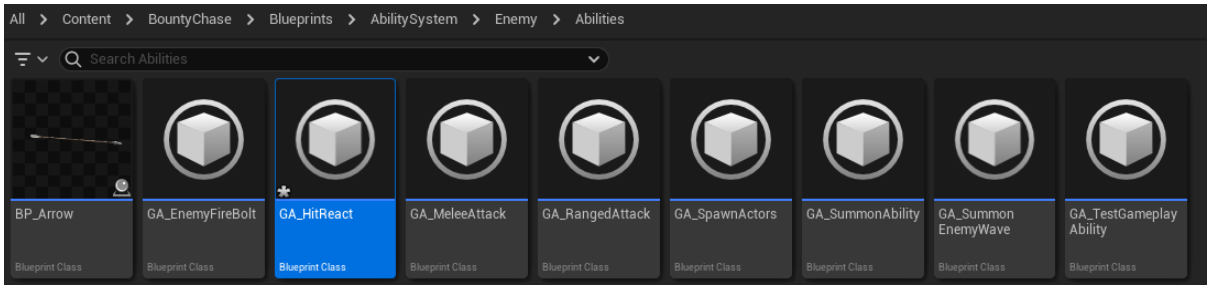


Рисунок 4.1 – Приклади заклять закріплених за ворогами

Таким чином гравець не може рухатись чи застосовувати закляття поки він має геймплейну мітку приголомшення. Окрім цього, завдяки цій системі ми можемо використовувати різні мітки і створенні ефекти повторно для різних заклять з різною величиною, тобто ефект може тривати більше, або задіяти більше шкоди, або через певний час і все це задається у кожному окремому заклятті, залежно від ролі цього закляття у грі, дозволяючи легко комбінувати багато ефектів у ще більше різних заклять без додаткових складнощів, бо усі закляття вже включають в себе ефекти та атрибути з якими повинні проходити маніпуляції.

Одним з яскравих прикладів заклять є здібність реакції на отримання шкоди ворогами (рис. 4.2). При отриманні шкоди ворогом, ми отримуємо підтвердження, що ворог є частиною інтерфейсу “CombatInterface” і отримуємо монтаж реакції на шкоду і застосовуємо ефект приголомшення на період за замовченням, після чого завершуємо здібність, для того, щоб вона не викликати витік пам’яті.

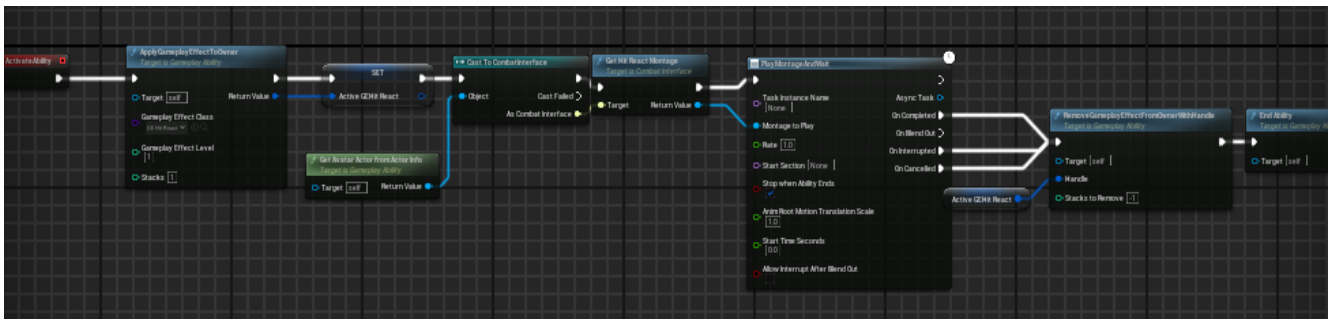


Рисунок 4.2 – Імплементація здібності “GA_HitReact”

Інші здібності застосовуються подібним чином, лише дечим змінюючи умови за яких виникають ті, чи інші ефекти, наприклад застосування снарядів вимагає

прорахунків фізичного компоненту, для перевірки того факту, що снаряд попав у іншого персонажа, що є частиною геймплейної системи і є ворогом або другом. Такі здібності вимагають використання інших частин системи Unreal Engine 5, що не ускладнює логіку, бо ми використовуємо вже завчасно продумані компоненти, що виконують роботу за нас. Одного з таких акторів, що застосовує фізичний компонент можна побачити на рисунку 4.3.

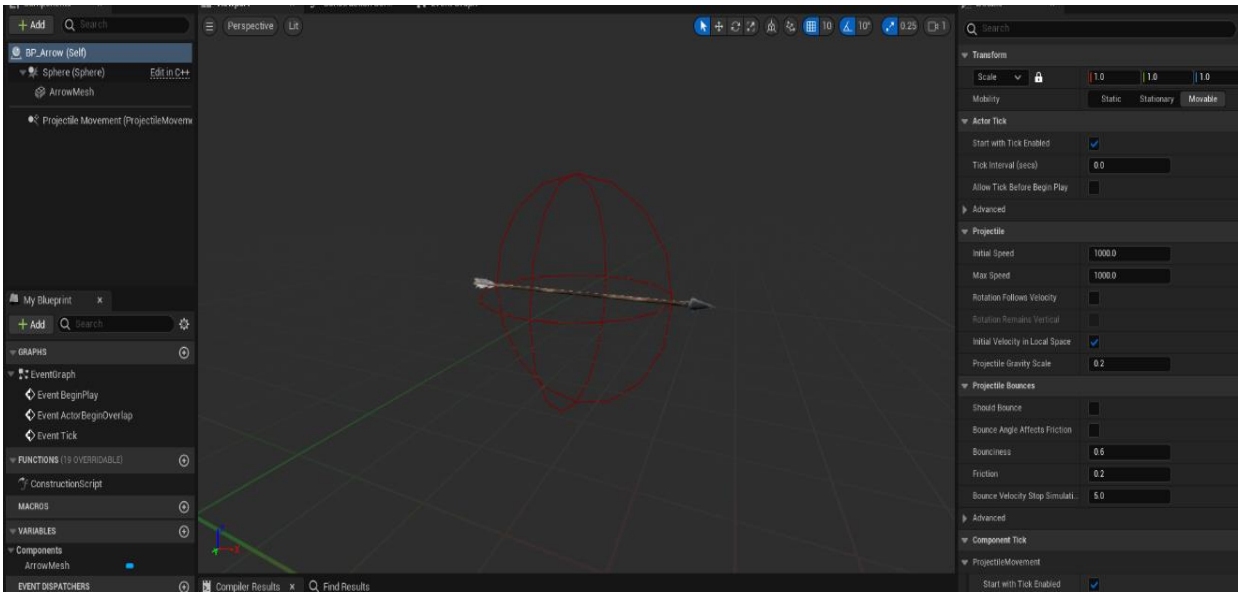


Рисунок 4.3 – Актор стріли, який застосовує фізичний компонент

Таким чином, ми можемо побачити ефективність даної системи не тільки у її застосуванні, а й у тому, щоб масштабувати цю систему у подальшому і створювати нові ефекти і закляття.

4.2 Система високоякісних тіней

Запропонована система застосовує метод плиткової відкладеної тіні, а також тіньові карти. Протягом п’яти етапів система здатна створити тіні, які майже не

поступаються у якості тіням влаштованим у систему Unreal Engine 5, а також генеруються швидше за ті, що створені ігровим рушієм самостійно.

Було прийняте рішення використовувати індекс акторів, що мають відображати тінь і виділяти тіньову карту для кожного з них окремо, аніж перенавантажувати систему і створювати одну тіньову карту, що має бути створена для кожного джерела світла та зберігати інформацію про тіні, що стосуються усіх об’єктів на сцені, а не тільки на необхідних акторів (рис. 4.4).

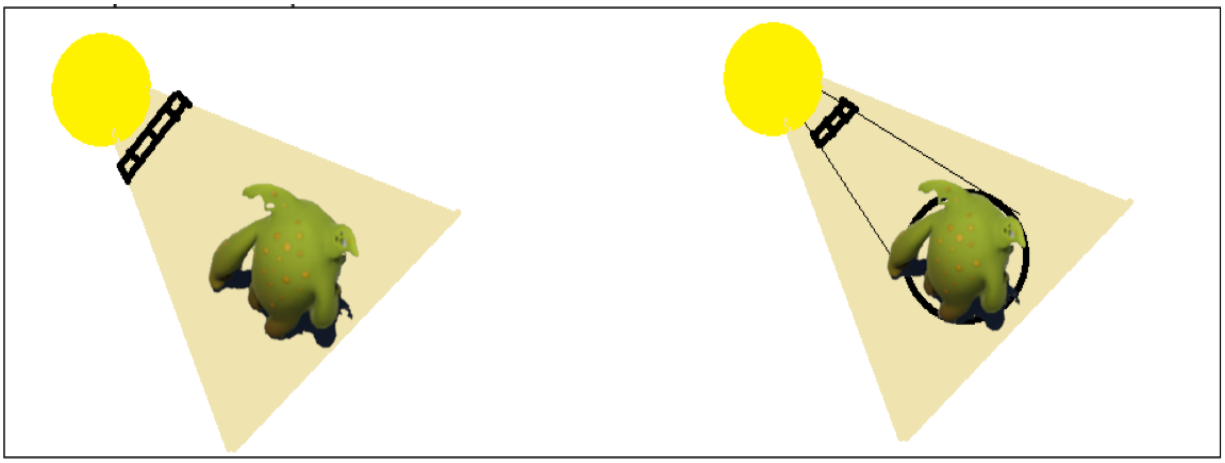


Рисунок 4.4 – Порівняння стандартного алгоритму (зліва) з першим етапом запропонованого алгоритму (справа)

Друге не дає змогу використати переваги статичних об’єктів і того факту, що вони не впливають на тінь у сцені. Після цього ми робимо заклик до малювання у приближену тіньову карту, що зберігає нам пам’ять. Результат другого етапу переноситься у етап зважження, де ми зменшуємо отриману тіньову карту у чотири рази, зменшуючи ширину та висоту вдвічі. У наступному етапі переглядається статус тіні, та чи є вона повною, відсутньою чи частковою, де перші два етапи одразу переносять нас у кінець алгоритму. Останнім етапом є перенесення, де результати часткових тіней додаються до фінальної тіньової карти, де знаходяться усі об’єкти і це зберігає нам деяку потужність і збільшує якість.

4.3 Система штучного інтелекту

Система штучного інтелекту наразі включає в себе три основних компоненти: дерево поведінки (“BehaviorTree”), систему взаємодії з середовищем (“EQS”), та завдання (“Tasks”). Дерева поведінки відповідають за повну логіку поведінки ворога (рис. 4.5).

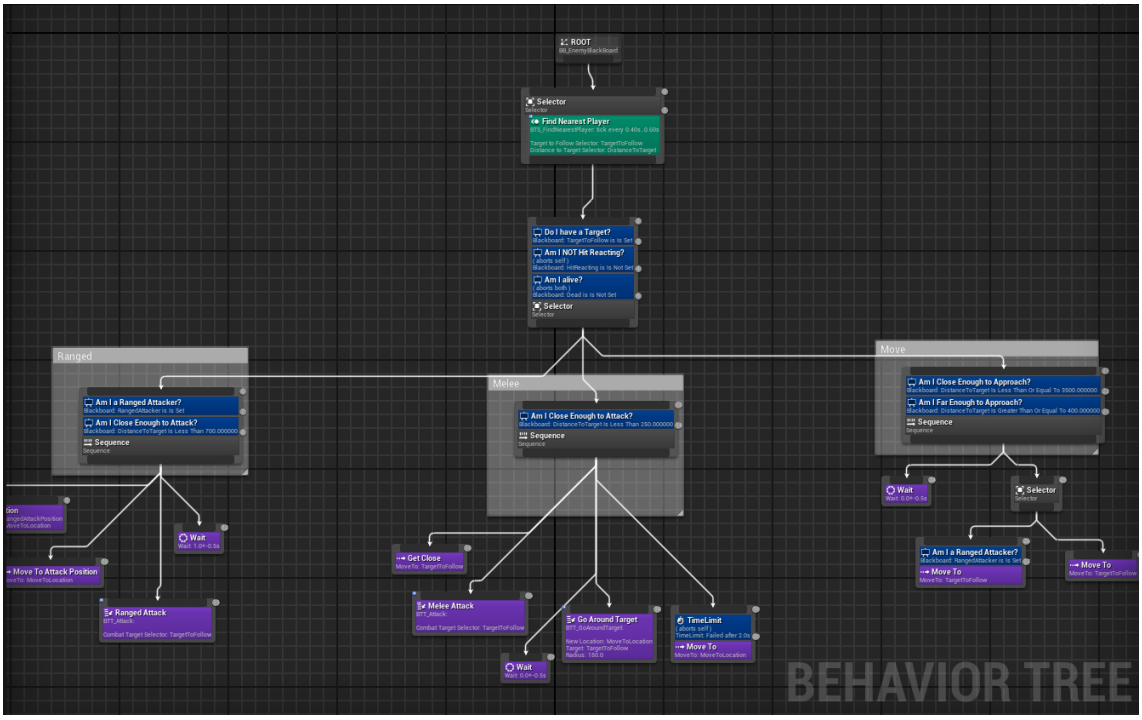


Рисунок 4.5 – Поведінка звичайного ворога зображена у дереві поведінки

У цьому дереві ми можемо побачити застосування ворогом завдань, наприклад пошуку гравця, очікування, переміщення до іншої позиції, або використання заклять для нанесення шкоди гравцю. Більшість з цих дій потребують взаємодії з середовищем, що вимагає необхідної системи для аналізу середовища і перевірку виконання умов (рис. 4.6).

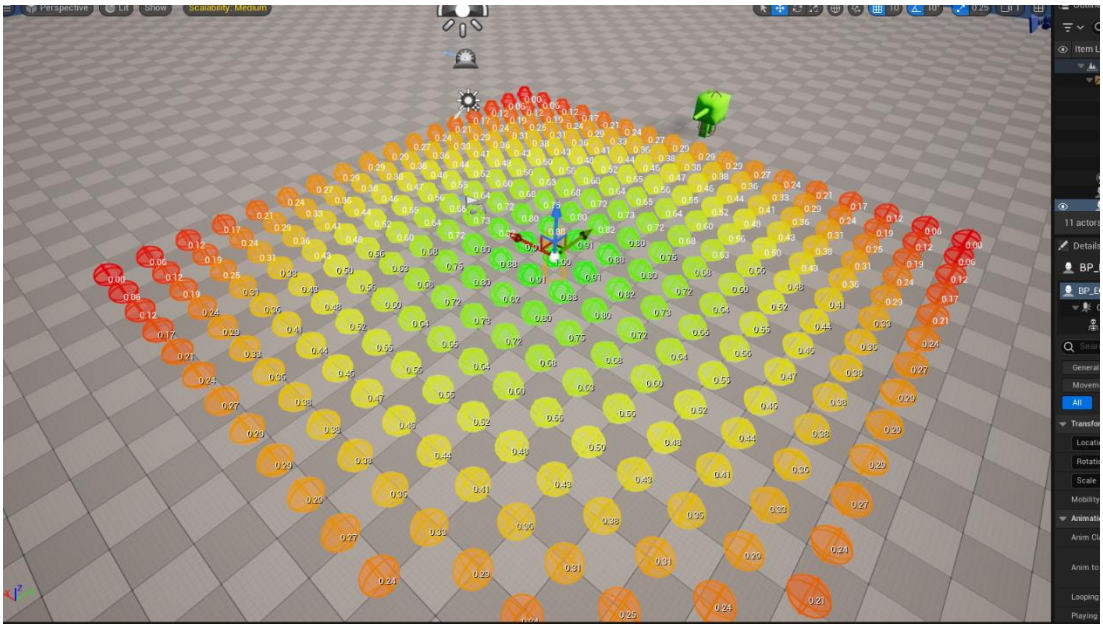


Рисунок 4.6 – Приклад перевірки застосування системи аналізу середовища

У такій перевірці застосовується багато чинників, багато з них пов’язані з внутрішніми системами штучного інтелекту UE5, наприклад можливість пересуватись по поверхні, що задається компонентом NavigationMesh. Завдяки цьому система легко прораховує найближчу дистанцію між акторами та здатна зробити оцінку найефективнішої позиції у якій потрібно знаходитись для атаки. У даному випадку розглянуто атаку ворога, що здатен застосовувати закляття з відстані, умовою якого є не бути занадто далеко, не потребувати зайвих переміщень, а також здатність побачити гравця із зазначеної позиції, на основі цих факторів створюється рівняння, яке робить внесок у вартість позиції, де менша вартість означає гіршу позицію, а більша вартість кращу позицію. Також ми можемо одразу відсікти неправильні позиції, зробивши вартість нулем, якщо неможливо побачити ворога з неї (рис. 4.7).

Це забезпечує повторне використання вже створених завдань та системи взаємодії з середовищем задля легкого впровадження у різну поведінку для різних типів ворогів, враховуючи, що вони можуть вимагати різних умов для створення необхідних кроків і не робити гру занадто простою для гравця.

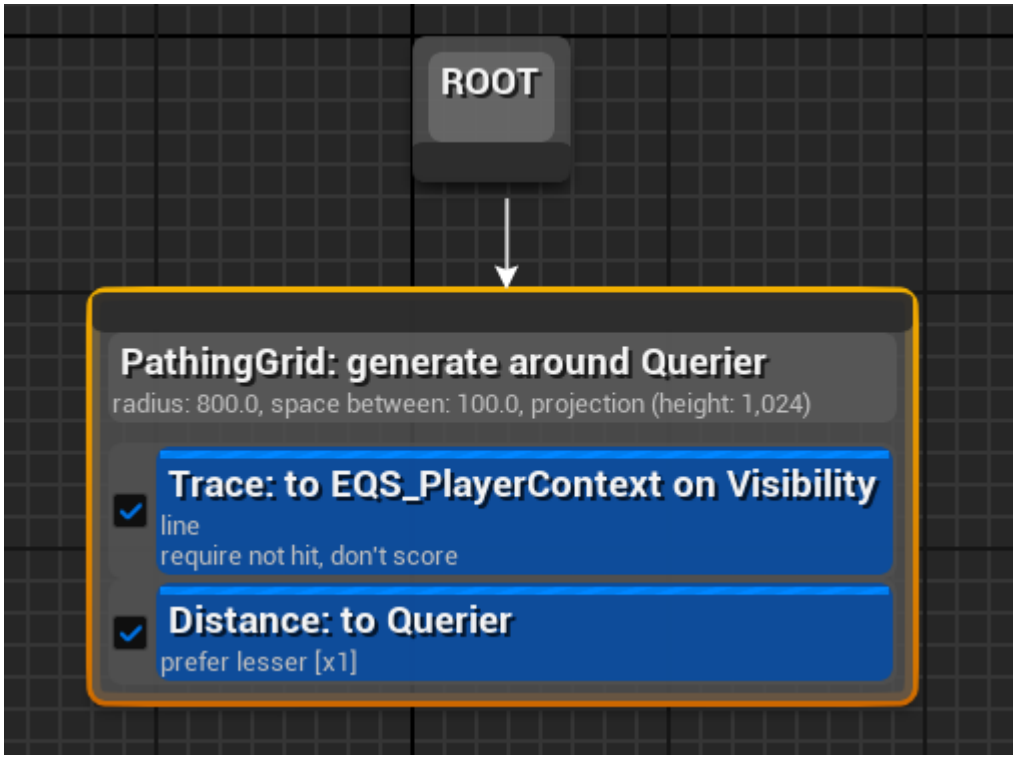


Рисунок 4.7 – Оцінка можливості побачити ворога (гравця)

Таким чином ми можемо побачити легкість застосування штучного інтелекту для створення гри, що можна буде розширити у поточному вигляді без зайвих складнощів.

5 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Дійсно, на кожному етапі розробки програмного забезпечення важливо проводити його тестування. Цей процес не лише підтверджує правильність роботи системи, але й допомагає виявити і усунути помилки та недоліки. Тестування має бути постійним і систематичним, починаючи з моменту розробки і до фінального випуску продукту. Особливу увагу варто приділяти тестуванню ігрових застосунків, оскільки вони мають свої особливості і вимоги до якості. Такий підхід дозволяє забезпечити користувачам максимальний комфорт та задоволення від гри. Проте, хоча тестування ігрових програм може відрізнятися від стандартного тестування програмного забезпечення, основні принципи залишаються незмінними.

Як правило, для проведення тестування програмного забезпечення, включаючи ігрові застосунки, формується окрема команда тестувальників, яка відповідає за якість продукту. Ця команда складається зі спеціалістів з різних областей, включаючи тестувальників, аналітиків та інженерів з якості. Вони розробляють тестову документацію, яка включає в себе плани тестування та набори тестових кейсів, а також відстежують і документують виявлені дефекти. Крім цього, команда тестувальників відіграє ключову роль у керуванні процесом якості розробки продукту. Таким чином, тестування стає не лише складовою частиною розробки програмного забезпечення, але і гарантією його якості та надійності.

У ході розробки програми у нашому випадку кожен розробник проводив тестування тієї частини, над якою він працює сам, тобто кожна частина системи була завчасно продумана її тестувальником. Основними тестами були мануальні тести, а також часто використовувався підхід чорної і білої скриньки. Отже, для виявлення помилок було використано два основних підходи. По-перше, тестувалося програмне забезпечення, керуючись вимогами до застосунку, що означає, що було перевірено, як програма працює з точки зору кінцевого користувача. Цей підхід дозволяє ідентифікувати будь-які аспекти програми, які

можуть вплинути на користувача під час її використання. Зокрема, було перевірено функціонал, доступний для користувача, та впевненість у його правильному виконанні. По-друге, знання коду застосунку дозволяло їм перевіряти моменти гри, які могли бути проблемними з точки зору програмного коду. Це включало аналіз алгоритмів, що використовувалися в грі, та перевірку їхньої логіки. Було уважно перевірено, чи передбачено всі можливі сценарії взаємодії з гравцем, і виявляли ті моменти, коли алгоритми не забезпечували правильну реакцію на певні дії гравця. Такий аналіз допомагав виявляти й виправляти потенційні проблеми з логікою гри, що сприяло підвищенню її якості та коректності.

Отже, процес тестування передбачав проведення ігрових сесій, в ході яких було активно використано усі можливі функції системи, перевіряючи їхню роботу та спостерігаючи за її поведінкою. Кожен аспект гри аналізувався з увагою до деталей, з метою виявлення будь-яких можливих недоліків або помилок. Після виявлення недоліків у роботі програми, було складали баг-репорти. Ці документи містили детальний опис проблеми, умови, за яких вона виникла, кроки для її відтворення та очікуваний результат. Такий підхід до документування дефектів дозволяв зберегти якість і надійність програмної системи під час її розвитку та після випуску на ринок. Приклади тестових випадків наведено у таблицях 5.1 - 5.3, що було створено у процесі розробки програмної системи.

Таблиця 5.1 – Тестовий випадок неправильного оновлення здоров’я

Назва:	Неправильне оновлення здоров’я
Короткий опис:	При зменшенні максимального здоров’я інтерфейс відображає неправильний колір наповнення здоров’я
Компонент програми:	Інтерфейс користувача
Пріоритет:	P3 Низький
Серйозність:	S5 Тривіальний
Кроки відтворення:	Прокачати здібність для зменшення сили. Використати її (зміниться атрибут максимального здоров’я) при здоров’ю близькому до максимального.
Очікуваний результат:	Прогрес бар здоров’я оновлюється градієнтом між відтінками оранжевого.
Фактичний результат:	Прогрес бар здоров’я оновлюється червоним кольором, повністю заповненим.

Таблиця 5.2 – Тестовий випадок вогню по своїм

Назва:	Вогонь по своїм
Короткий опис:	Вороги наносять шкоду один одному з деяких заклять
Компонент програми:	Геймплей. Закляття.
Пріоритет:	P2 Середній
Серйозність:	S3 Значний
Кроки відтворення:	Створити багато ворогів зі здібністю Firebolt. Зайти у радіус знаходження гравця до багатьох з них одразу. Створити ситуацію при якій один ворог стоїть перед іншим і атакує гравця. Дочекатися активації закляття ворогом
Очікуваний результат:	Закляття попадає у іншого ворога і зникає без побічних ефектів
Фактичний результат:	Ворог наносить шкоду іншому ворогу

Таблиця 5.3 – Тестовий випадок занадто швидких заклять

Назва:	Занадто швидкі закляття
Короткий опис:	Швидкі закляття проходять крізь гравця / інших ворогів
Компонент програми:	Геймплей. Закляття
Пріоритет:	P2 Середній
Серйозність:	S3 Значний
Кроки відтворення:	Надати гравцю здібність Fireblast. Використати у бік ворогів
Очікуваний результат:	Закляття попадає у ворога чи перший об’єкт з яким зустрічається.
Фактичний результат:	Закляття проходить крізь ворогів та інші об’єкти.

Більшість інших тестувань застосовувалося у процесі написання програми як за допомогою Blueprints, так і за допомогою мови C++. Такі способи включали перевірку усіх доступних гравцеві механік та здібностей, ефектів, чи інших частин гри.

6 ВПРОВАДЖЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

6.1 Наукове впровадження проєкту

Враховуючи тему кваліфікаційної роботи було створено наукові тези для форуму, що проходив з 16 по 18 квітня 2024 року в дистанційному режимі на основі ХНУРЕ, а саме XXVIII Міжнародний молодіжний форум "Радіоелектроніка та молодь у XXI столітті".

“ОПТИМІЗАЦІЯ АЛГОРИТМУ ПЛИТКОВОГО РЕНДЕРІНГУ І ТІНЬОВИХ КАРТ” – стаття, що була опублікована у належному збірнику конференції №6 [11].

У ній представлена основна ідея алгоритму генерації тіней, показано і порівняно ефективність з Epic тінями в UE5, алгоритм детально описано і зазначено завдяки чому досягається ефективність, не втрачаючи у той самий час якість наведених тіней.

6.2 Практичне впровадження проєкту

На основі матеріалів кваліфікаційної роботи була представлена участь у виставці технічної творчості молоді 2024 року. Проект, що був представлений і захищений під час виставки, а саме “Ігровий програмний застосунок у жанрі Roguelite RPG. Bounty Chase”, у якому гравцеві було поставлено за основну мету пройти максимально можливу кількість хвиль ворогів, між якими він здатен покращувати свої атрибути та здібності, у той самий час відкриваючи нові. Після поразки гравець повертався сильнішим, з поліпшеними здібностями, і міг пройти гру знову. З кожним рівнем вороги ставали сильнішими, а кілька гравців могли одночасно проходити один і той самий рівень.

Переваги розробленого проекту включали кросплатформенність, простий та зрозумілий інтерфейс, легке впровадження нових значень дизайнерами за допомогою таблиць, а також підтримку мультиплеєрної гри. Ці функції сприяли зручності використання та розширенню можливостей проекту, забезпечуючи цікавий та захоплюючий геймплей для гравців різного рівня.

Виставка мала місце з 16 по 18 квітня 2024 року у ХНУРЕ і проводилася в дистанційному режимі [12].

ВИСНОВКИ

Під час виконання складової частини наданої комплексної роботи на тему "Ігровий програмний застосунок у жанрі Roguelite RPG: Ігрові механіки, 3D графіка, AI" було проведено аналіз предметної галузі, досліджено як основних конкурентів у цьому сегменті, так і ігрові застосунки, що мають деякі девіації за жанром і реалізацією унікальних механік. Виокремивши переваги та недоліки кожного з конкурентів, їх було враховано у процесі проектування зазначеного застосунку.

У подальшому робота включила постановку завдання, що охоплювало розробку ворогів, системи заклять та їх функціонування у мультиплеєрному режимі. Було визначено, які саме закляття мають бути, щоб найефективніше доповнити ігровий світ і створити ворогів, що найкраще відповідають концепції гри.

Інший етап роботи включав детальне проектування інтерфейсу користувача (UI/UX). Було створено прототипи інтерфейсу гри та детально розглянуто інтерфейси для геймплею з основною інформацією. Також було спроектовано інтерфейс для використання механіки заклять.

Окремої уваги заслуговує розробка частини проекту, а саме системи заклять. Було створено візуальні ефекти, розроблено компоненти для Unreal Engine, що відповідають за застосування заклять, а також систему динамічних тіней та штучний інтелект, який може використовувати закляття та реагувати на дії гравця відповідно до актуальної інформації. Також були створені компоненти Unreal Engine для взаємодії між закляттями, гравцями і ворогами, а також їх атрибутами, які є основою для всього геймплею.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Utkin Y. E. Automated tile texturing in unity tileset map / Y. E. Utkin // Радіоелектроніка та молодь у XXI столітті : матеріали 27-го Міжнар. молодіж. форуму, 10–12 травня 2023 р. – Харків : ХНУРЕ, 2023. – Т. 6, ч. 1. – С. 349–350.

2. Szabados, György Norbert, et al. "Roguelike Games: The Way We Play." International Journal of Engineering and Management Sciences 7.4 (2022): 80-92.

3. Arenas, Daniel Luccas, Anna Viduani, and Renata Brasil Araujo. "Therapeutic use of role-playing game (RPG) in mental health: A scoping review." Simulation & Gaming 53.3 (2022): 285-311.

4. Pangilinan, Stephen E., and Joshua Whorton. "Exploring Non-Player Character Types in Games: Enemies." (2023).

5. Supergiant’s Games Hades URL: https://www.supergiantgames.com/games/hades/ (дата звернення 12.05.2024)

6. The Binding of Isaac (videogame) URL: https://en.wikipedia.org/wiki/The_Binding_of_Isaac_(video_game) (дата звернення 12.05.2024)

7. Lumen. Unreal Engine 5 URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/lumen-global-illumination-and-reflections-in-unreal-engine?application_version=5.0 (дата звернення 11.05.2024)

8. Nanite. Unreal Engine 5 URL: https://www.unrealengine.com/en-US/blog/understanding-nanite---unreal-engine-5-s-new-virtualized-geometry-system (дата звернення 11.05.2024)

9. Kryvoshei, Artem. "GAME ENVIRONMENT MONSTER CHARACTER SYSTEMS." Сучасні комп’ютерні та інформаційні системи і технології: матеріали II Всеукраїнської наук.-практ. інтернет-конф. (01-12 грудня 2021 р., м. Мелітополь) / ред. кол.: В.М. Кюрчев, О.А. Єременко, С.В. Шаров та ін. Мелітополь: ТДАТУ, 2021. 175 с.: 67.

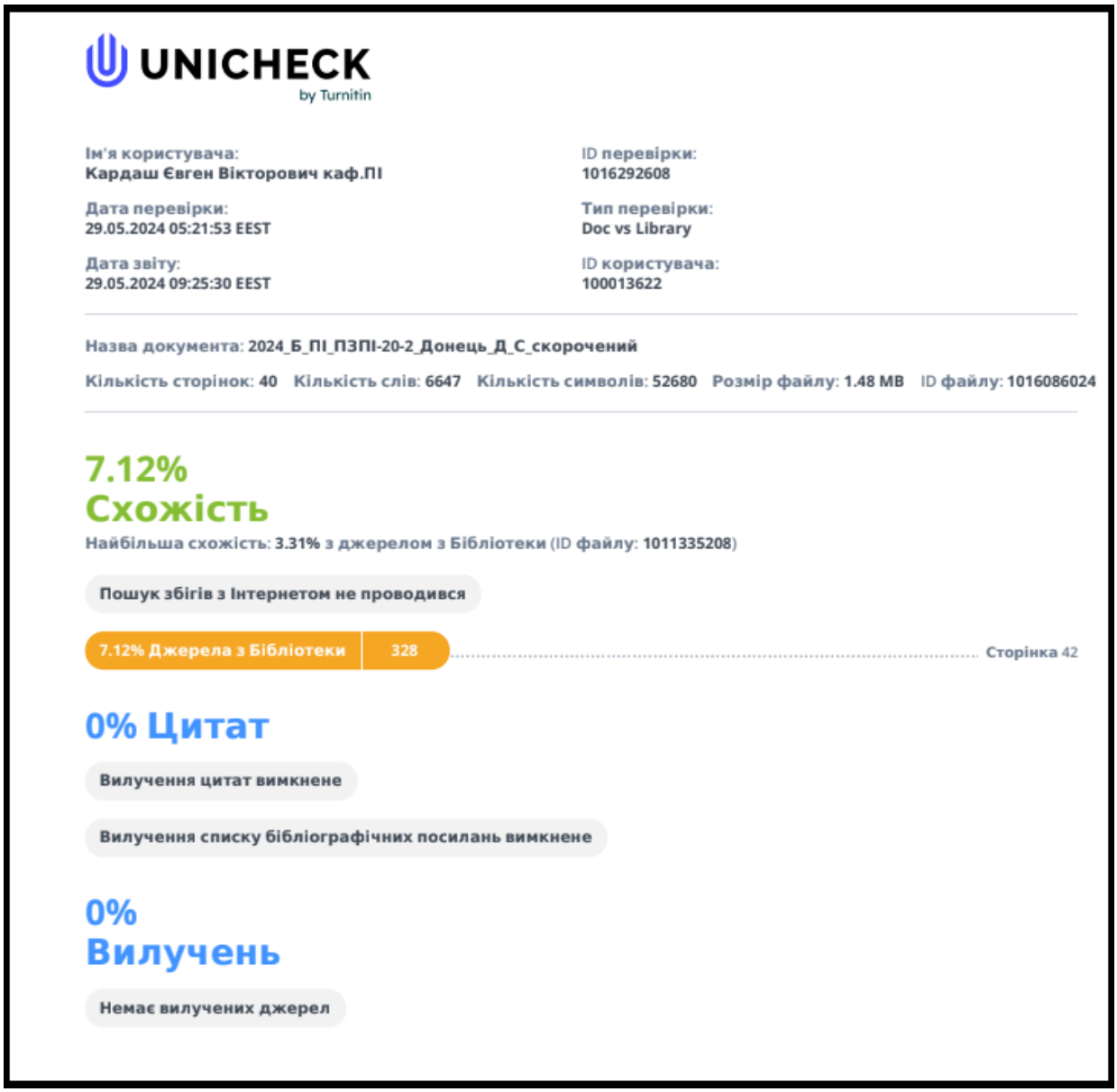
10. Kristiadi, Dedy Prasetya, et al. "The effect of UI, UX and GX on video games." 2017 IEEE International Conference on Cybernetics and Computational Intelligence (CyberneticsCom). IEEE, 2017.

11. Т. 6. Конференція «Комп'ютерної інженерії та захисту інформації». URL: https://openarchive.nure.ua/handle/document/26351 (дата звернення: 17.05.2024).

12. Каталог виставки технічної творчості молоді [Електронний ресурс]. URL: https://openarchive.nure.ua/handle/document/26355 (дата звернення: 17.05.2024).

ДОДАТОК А

Звіт результатів перевірки на унікальність тексту в базі ХНУРЕ



UNICHECK
by Turnitin

Ім'я користувача:
Кардаш Євген Вікторович каф.ПІ

ID перевірки:
1016292608

Дата перевірки:
29.05.2024 05:21:53 EEST

Тип перевірки:
Doc vs Library

Дата звіту:
29.05.2024 09:25:30 EEST

ID користувача:
100013622

Назва документа: 2024_Б_ПІ_ПЗПІ-20-2_Донець_Д_С_скорочений

Кількість сторінок: 40 Кількість слів: 6647 Кількість символів: 52680 Розмір файлу: 1.48 MB ID файлу: 1016086024

7.12%
Схожість

Найбільша схожість: 3.31% з джерелом з Бібліотеки (ID файлу: 1011335208)

Пошук збігів з Інтернетом не проводився

7.12% Джерела з Бібліотеки 328 ..... Сторінка 42

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

ДОДАТОК Б

Слайди презентації

Рисунок Б.1 – Слайд 1

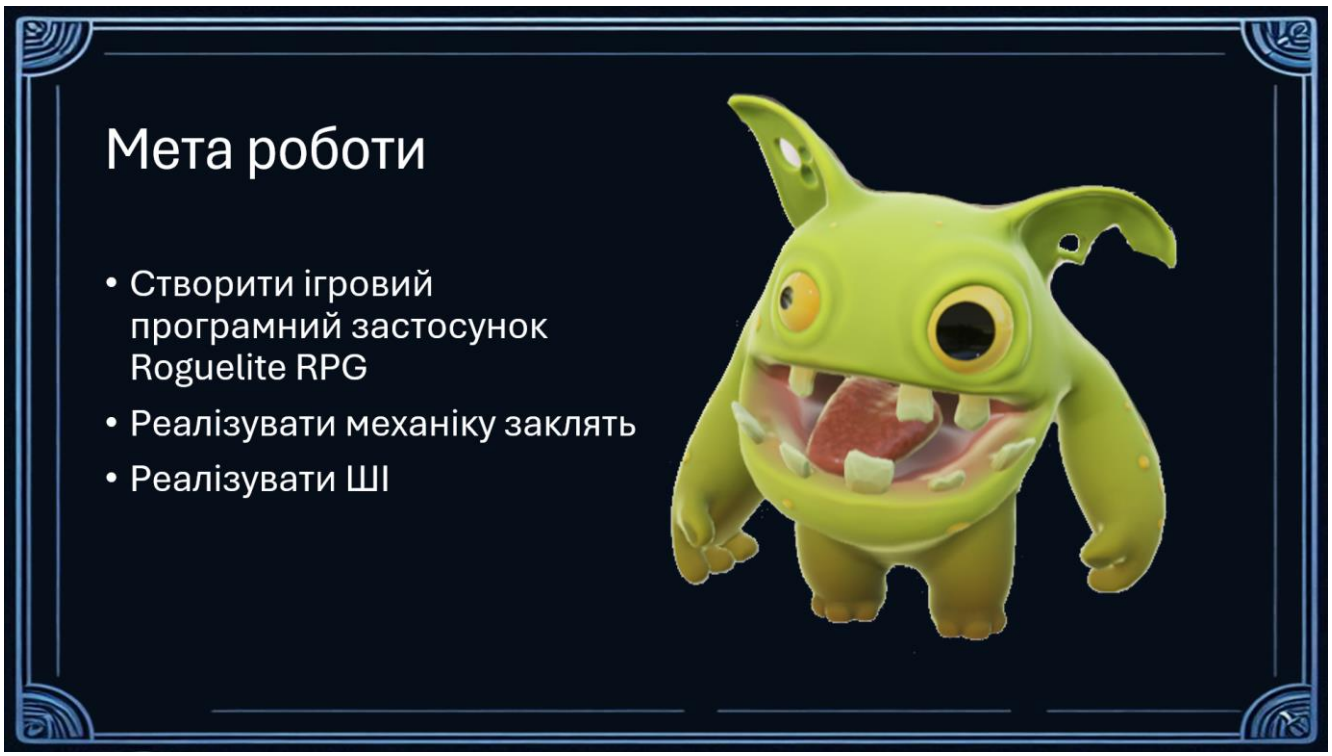


Рисунок Б.2 – Слайд 2

Рисунок Б.3 – Слайд 3

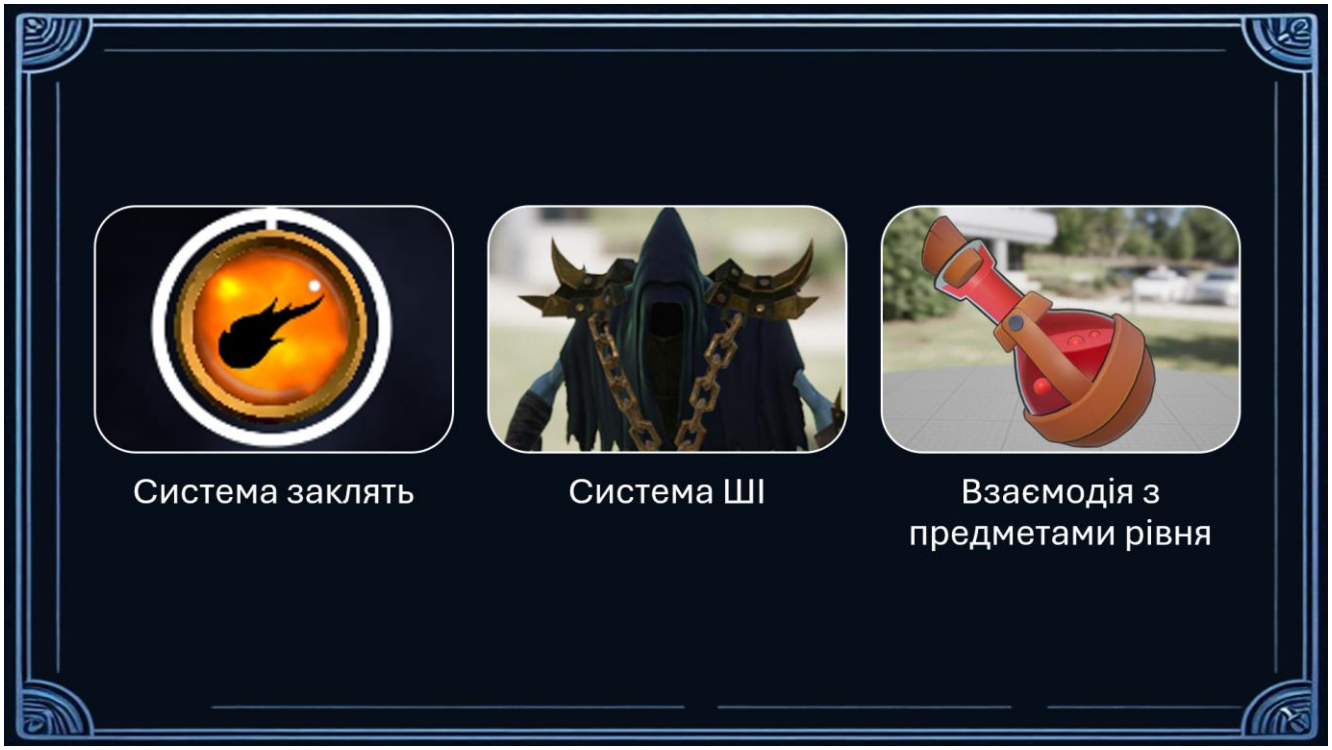


Рисунок Б.4 – Слайд 4

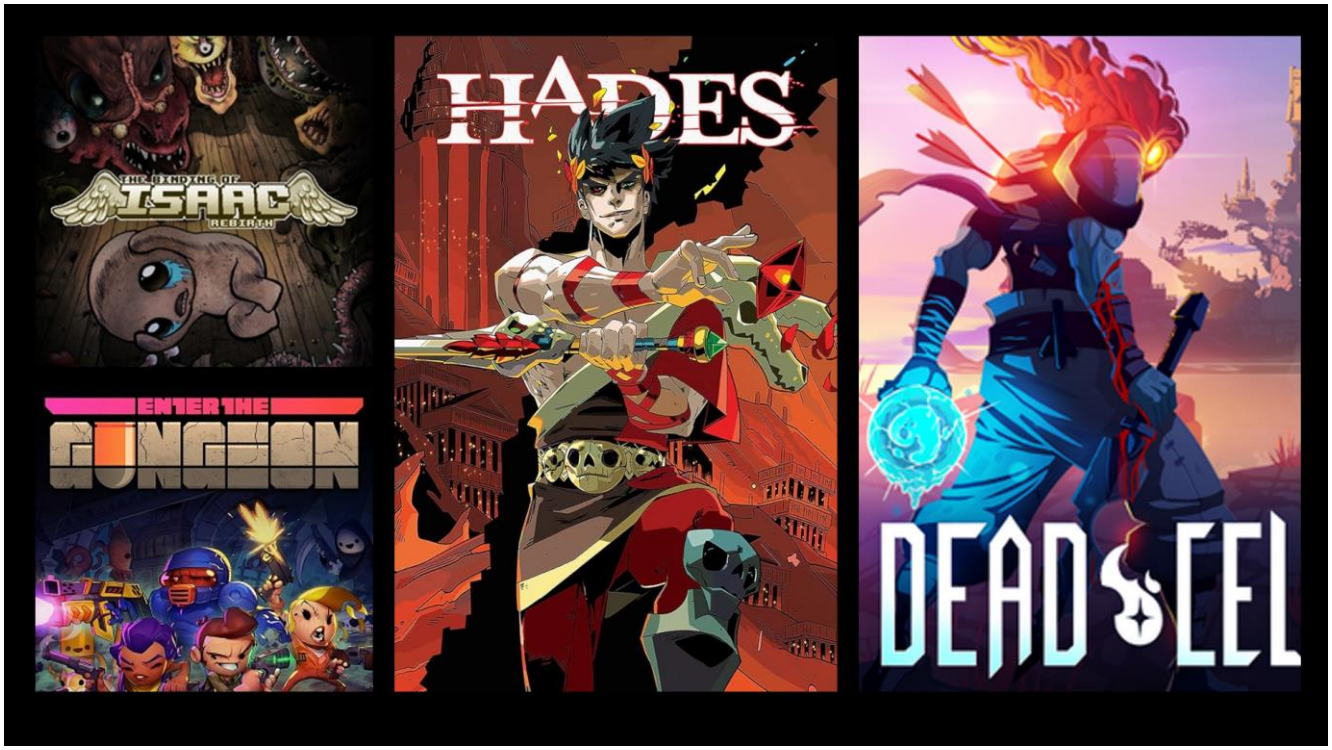


Рисунок Б.5 – Слайд 5

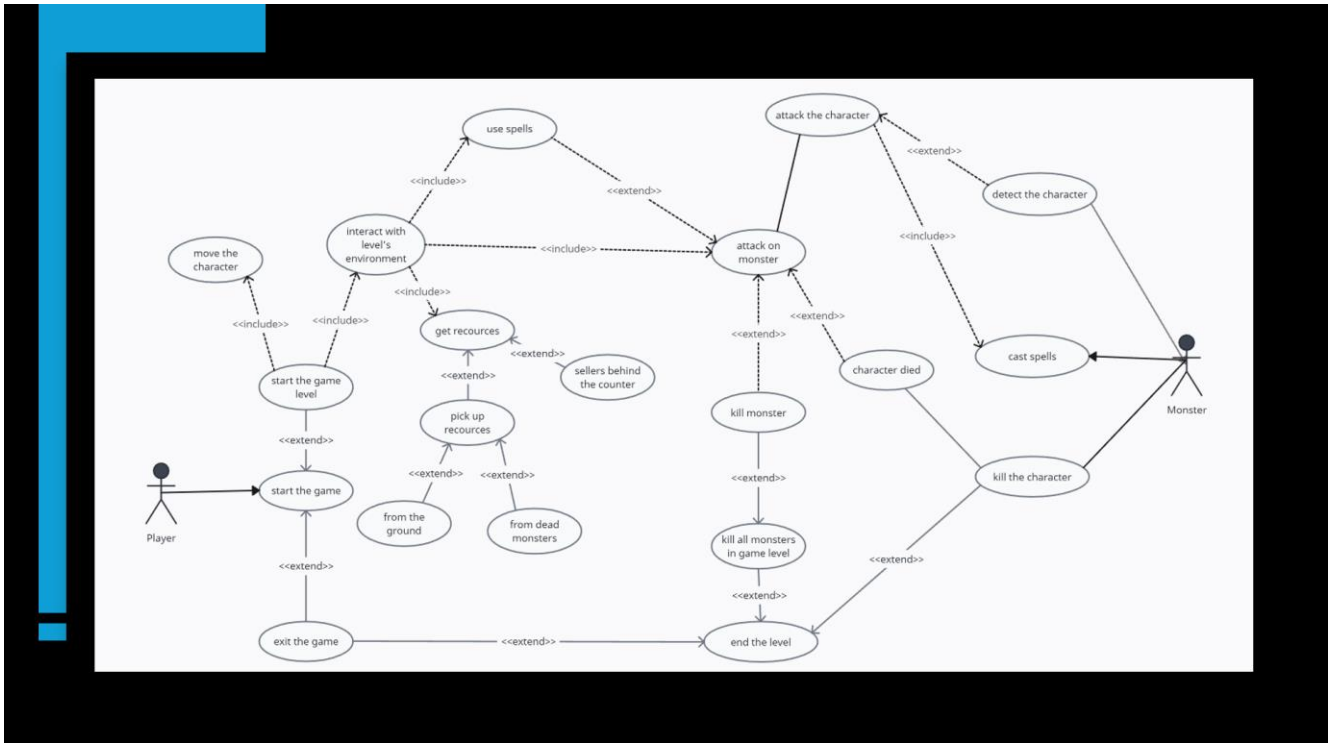


Рисунок Б.6 – Слайд 6

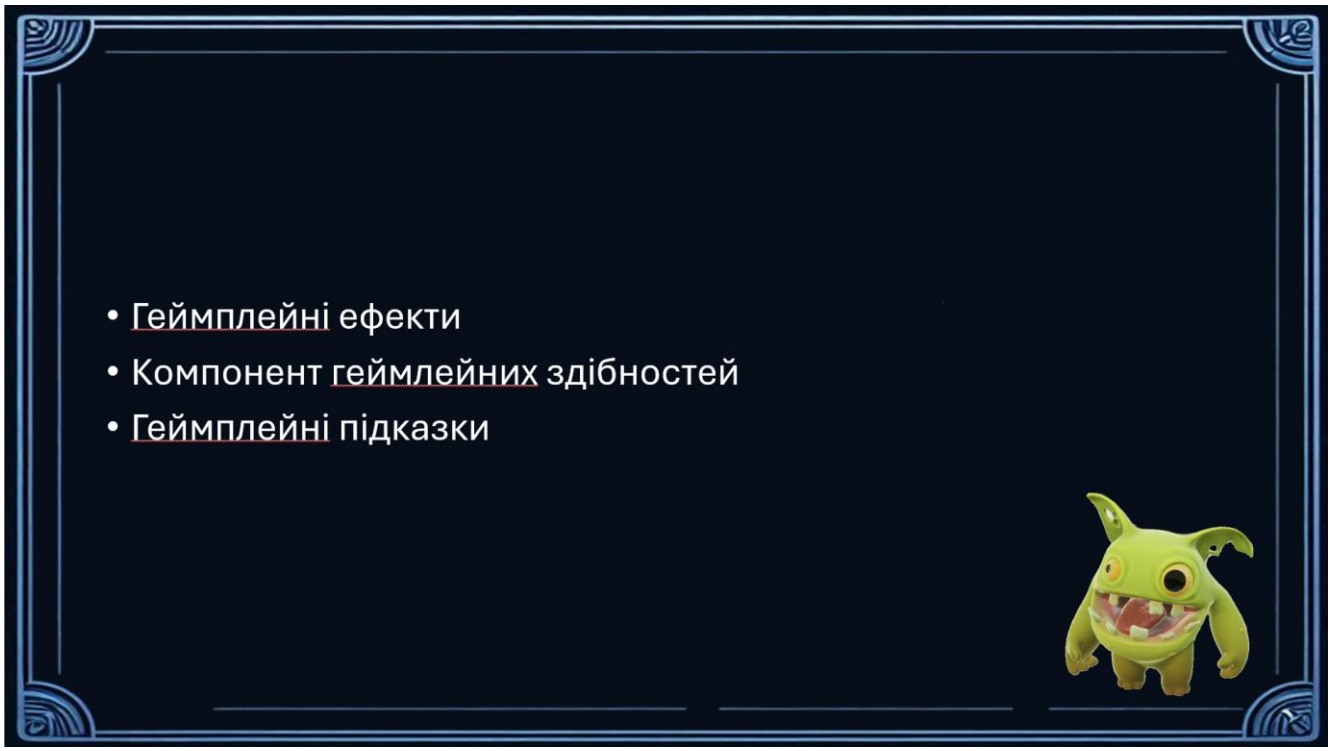


Рисунок Б.7 – Слайд 7

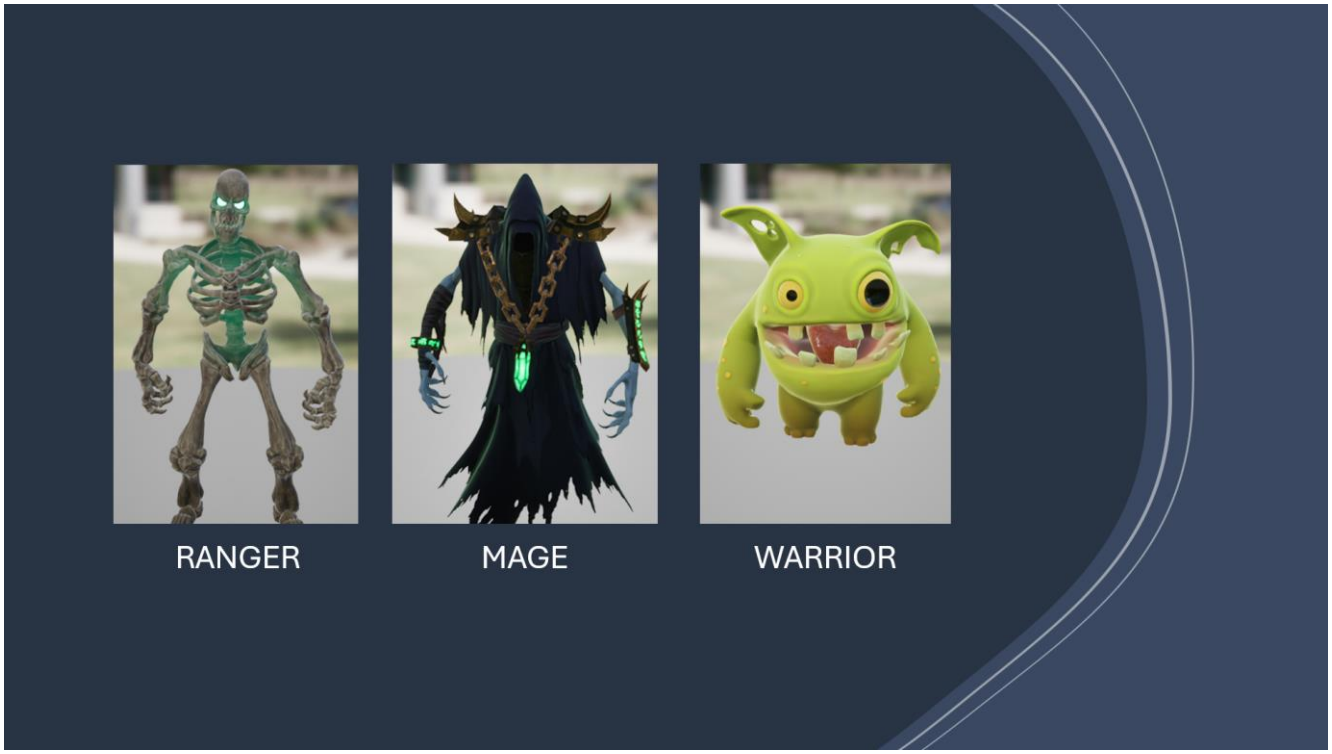


Рисунок Б.8 – Слайд 8

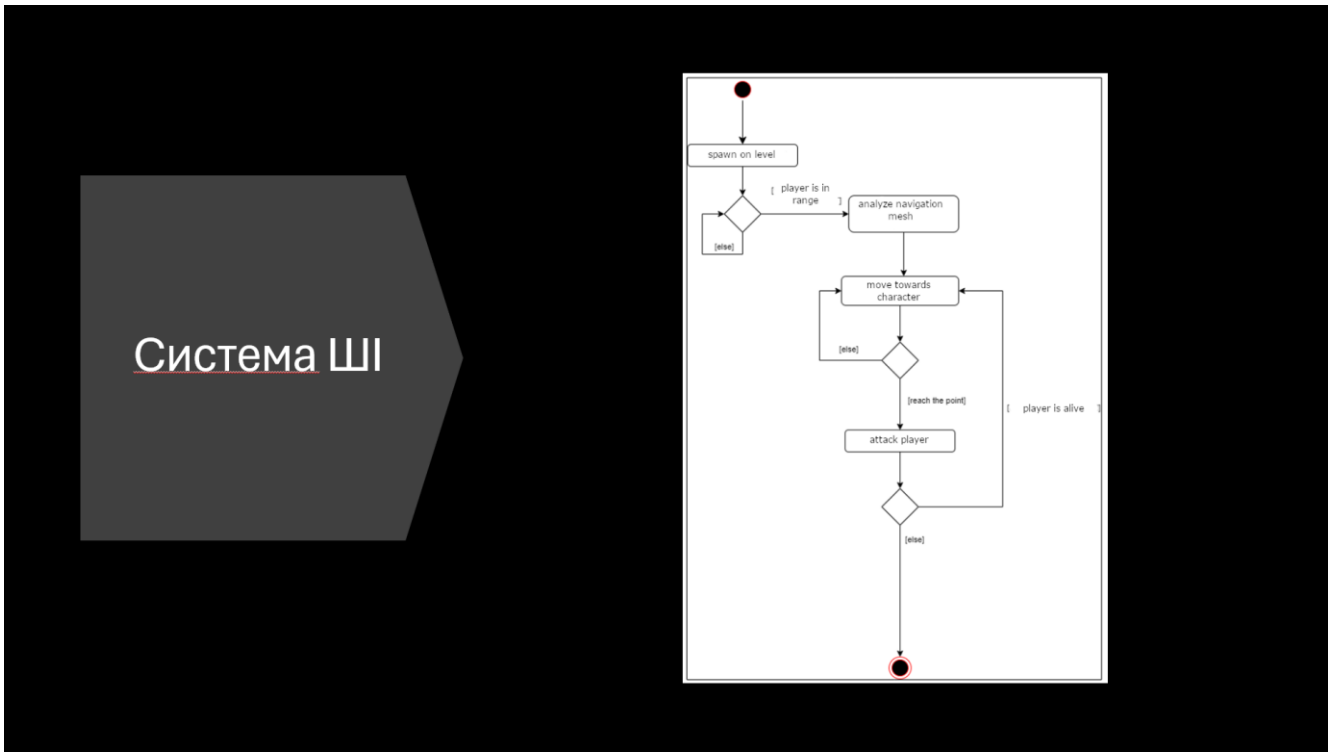


Рисунок Б.9 – Слайд 9

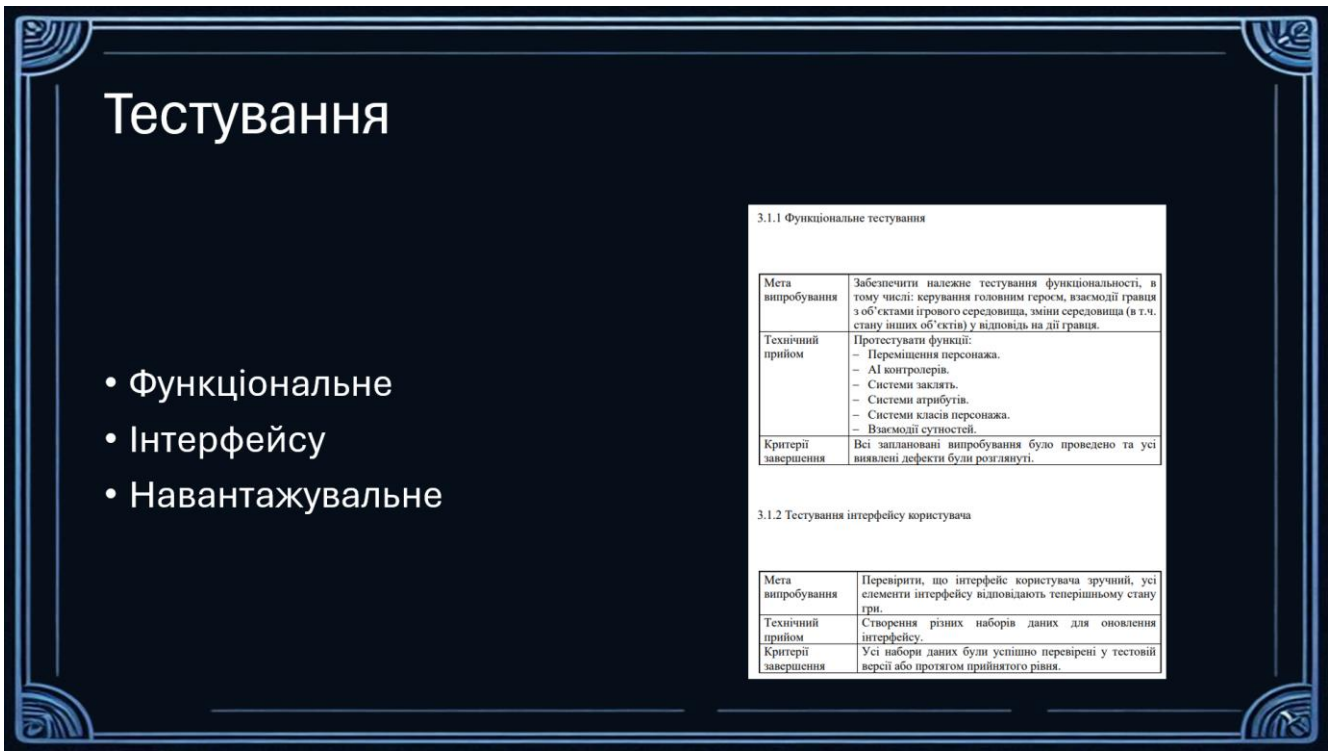


Рисунок Б.10 – Слайд 10

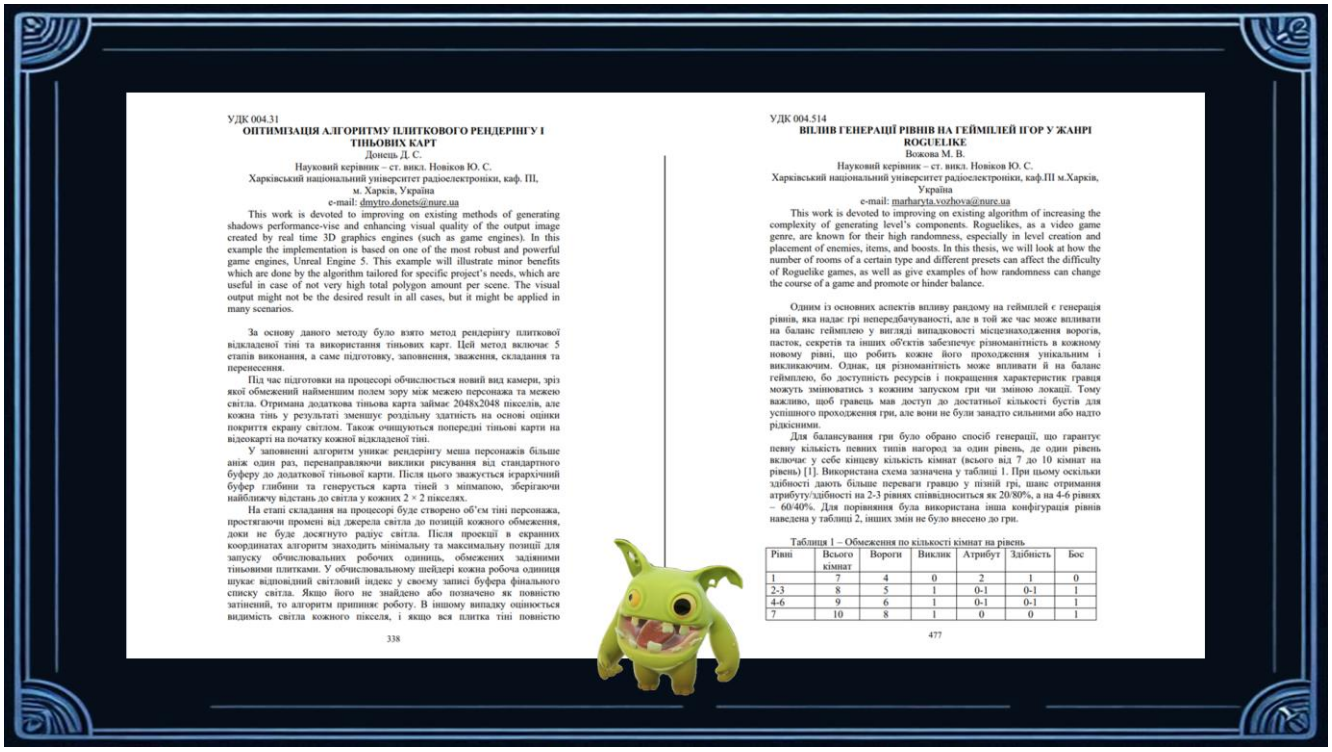


Рисунок Б.11 – Слайд 11

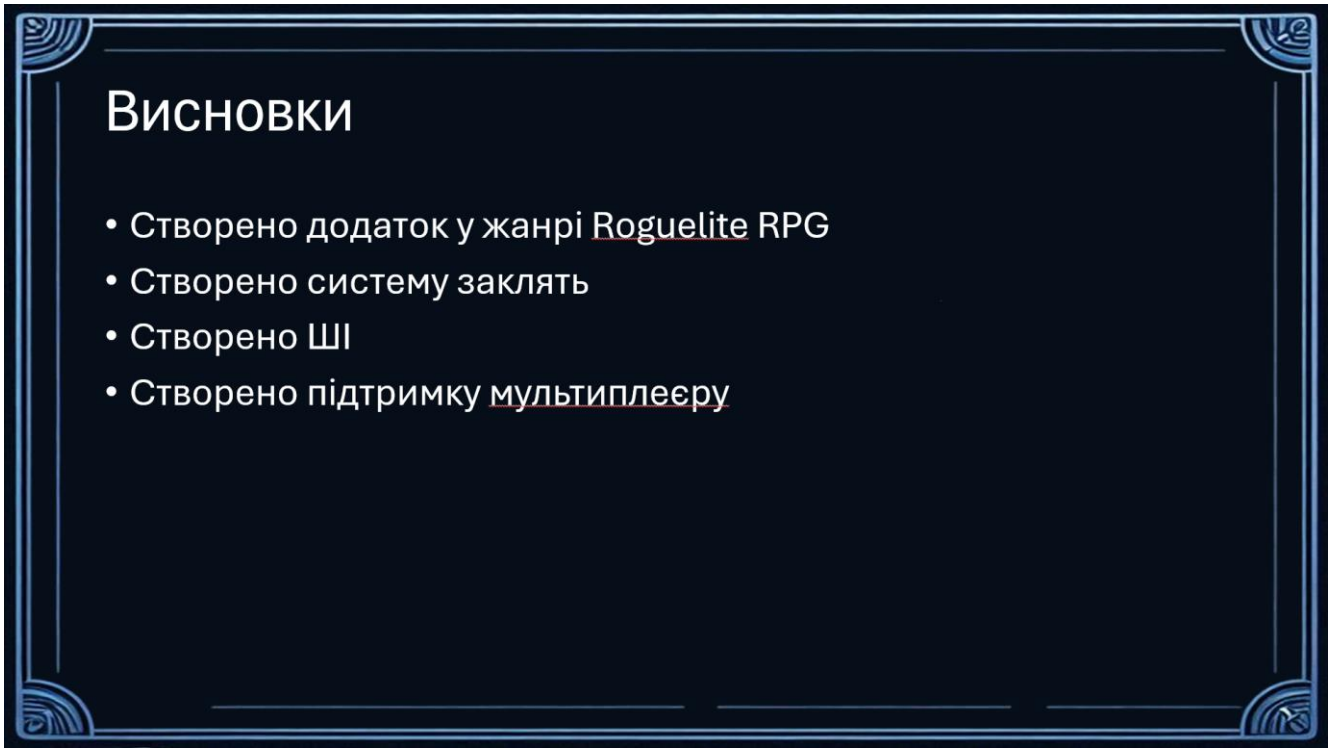


Рисунок Б.12 – Слайд 12

ДОДАТОК В

Геймдизайн-документ

Chapter 2. Game Description

2.1. Game Process Description

2.1.1. Objective

The main objective is to kill the final boss of the game with every god and perk combination available. Player can do so by going through levels of enemies until they reach the final boss level.

Global objectives include unlocking new gods and their perks and spells for future runs. During the run main objectives are destroying enemies, finding spells, dicey-upgrades, special currencies e.g. gold, saving characters and completing quests.

2.1.2. Game Flow Structure

Game has non-linear story. It doesn't mean it has different start or ending, but in-between may vary depending on triggered events. For example, if you find one character before freeing another, first will not mention the second one in dialogue. Game loop consists of multiple levels. Player always starts in Level 0, a.k.a. Lobby. It is non-combat room full of NPCs where player can modify his pre-run God Ability and Perks. Also they can unlock new Spells and Gods here. When player starts the run appears in Level 1. This level consists of combos of rooms, specifically Combat Room, Mini-Boss Room, Boss Room, Quest Room, Shop Room, Chill Room. It repeats for the next levels, 4 levels in total representing unique locations. Every Level ends with Boss Room. When player dies within any level he returns to Level 0 with some resources retained. Estimated Gameplay Duration 20-40 minutes, where every level consists of 8-10 rooms with some exceptions. Gameplay Duration by room is depicted on the table, where level are divided into parts due to rising difficulty.

Table 3. Gameplay Duration by Level

	Combat Room	Mini-Boss Room	Boss Room	Quest Room	Shop Room	Chill Room
Level 0	N/A	N/A	N/A	10-30s	N/A	10-30s
Level 1, Pt. 1	10-20s	N/A	N/A	15s	15s	10-30s
Level 1, Pt. 2	15-25s	20-30s	40-60s	15-20s	15s	10-30s
Level 2, Pt. 1	10-25s	25-35s	N/A	15-20s	15-20s	10-30s
Level 2, Pt. 2	17-30s	25-35s	50-70s	15-25s	15-20s	10-30s
Level 3, Pt. 1	15-25s	30-40s	N/A	15-20s	15-20s	10-30s
Level 3, Pt. 2	20-30s	35-45s	60-90s	15-20s	15-20s	10-30s
Level 4, Pt. 1	10-35s	40-45s	N/A	15-20s	15-20s	10-30s
Level 4, Pt. 2	N/A	N/A	90-120s	N/A	N/A	N/A

Рисунок В.1 – Опис ігрового проєкту з геймдизайн-документу

2.2.5. Status Effects

Status Effect is a temporary modification to the character, it may be both positive and negative, applied to enemies and player himself.

Bountiful: Drops additional gold on death, but be quick, it can expire!
Drunk: Move and attack slower.
Hex: Can't attack or use spells.
Immortal: Can't be slowed or damaged.

2.2.6. Passive Abilities

Passive Ability is a persistant status effect, usually more complex.

Thorny Faces: After you take damage, deal 30 damage to all enemies near you.
Golden Bubble: Absorb any instance of damage. Restores in 20 seconds.
Lucky Three: After you cast three spells ***Invoke***

2.2.7. Some Keywords

Anvil: Anvil falls on your opponents, dealing 100 damage in small area.
Bountiful: Drops additional gold on death.
Burn: Lose Dice Pact after use.
Choose One: Give a choice of three items. Player can select one of them.
Critical: Deal +150% damage.
Endless: Does not expire after exiting combat.
Gamble: Randomly choose between Positive and Negative effect.
Haste: Move and Attack 30% faster.
Hex: Can't attack or use spells.
Immortal: Can't be slowed or damaged.
Invoke: Cast random spell.
Refresh: Replenish your God's Call Charge or Spell Charge instantly.
Wind: Projectile that slows attack speed and move speed of your enemies by 30%.
Wither: Take 30 damage per second for 6 seconds.

2.2.8. Dice Rolls

Dice Roll is inspired by dash abilities in different games, however in game it looks like dice *rolls* at certain speed for defined distance. Speed and Distance may vary depending on status effects and passive abilities. Player has 3 charges to roll, they replenish after some time that player doesn't roll. Apart from this, player can cast *dice abilities* by rolling.

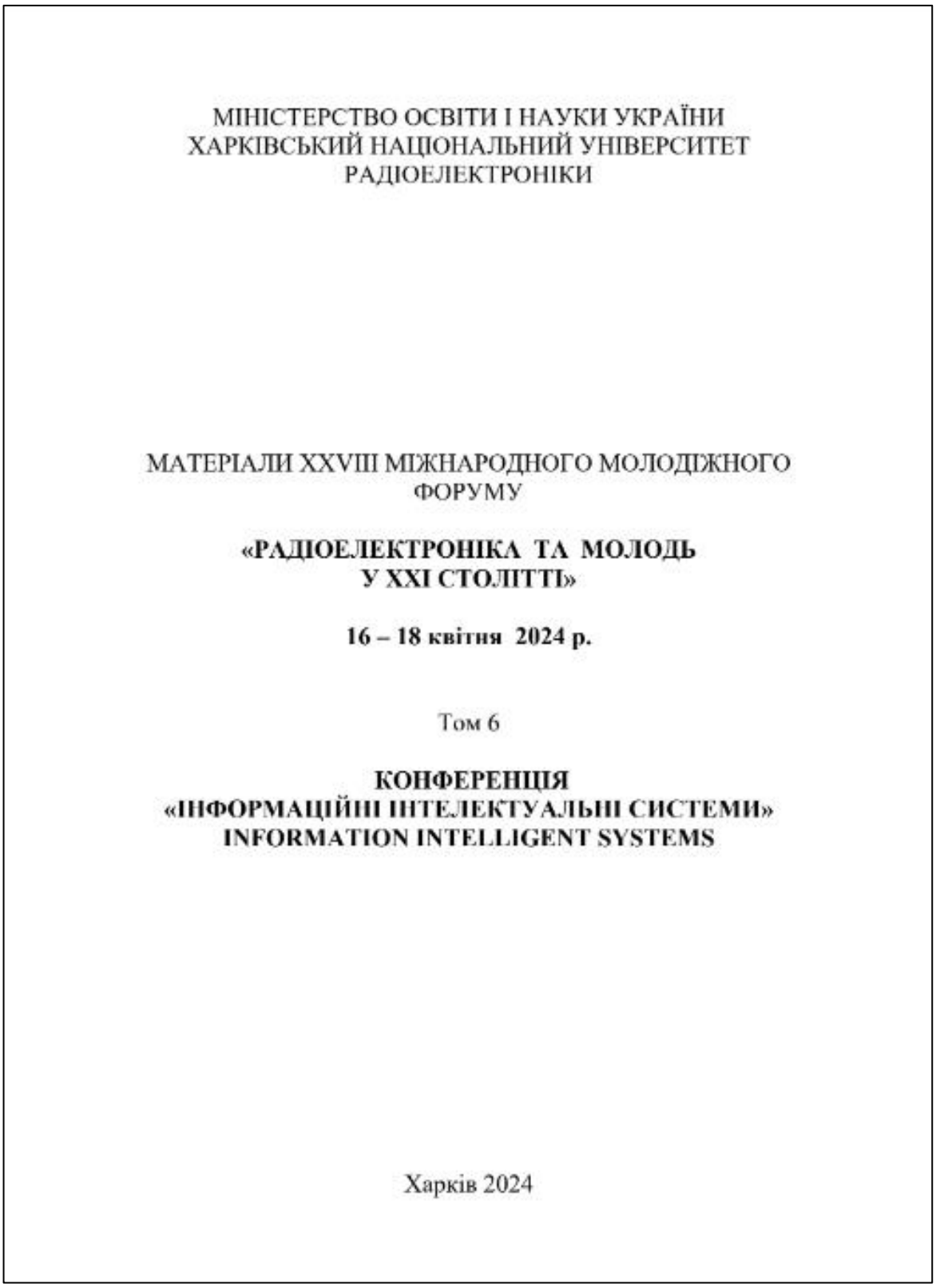
2.2.9. Dice Ability

Player casts dice ability by rolling 3 times in a row. The ability is determined based on combination of elements that were on dice' faces where player landed. If player lands on the face he has already

Рисунок В.2 – Опис основних ігрових механік

ДОДАТОК Г

Тези доповіді для науково-практичної інтернет-конференції



МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
РАДІОЕЛЕКТРОНІКИ

МАТЕРІАЛИ XXVIII МІЖНАРОДНОГО МОЛОДІЖНОГО ФОРУМУ

«РАДІОЕЛЕКТРОНІКА ТА МОЛОДЬ У XXI СТОЛІТТІ»

16 – 18 квітня 2024 р.

Том 6

КОНФЕРЕНЦІЯ
«ІНФОРМАЦІЙНІ ІНТЕЛЕКТУАЛЬНІ СИСТЕМИ»
INFORMATION INTELLIGENT SYSTEMS

Харків 2024

Рисунок Г.1 – Обкладинка збірника

Г.2 – Алфавітний покажчик збірника

УДК 004.31

ОПТИМІЗАЦІЯ АЛГОРИТМУ ПЛИТКОВОГО РЕНДЕРІНГУ І ТІНЬОВИХ КАРТ

Донець Д. С.
Науковий керівник – ст. викл. Новіков Ю. С.
Харківський національний університет радіоелектроніки, каф. ПІ,
м. Харків, Україна
e-mail: dmytro.donets@nure.ua

This work is devoted to improving on existing methods of generating shadows performance-vise and enhancing visual quality of the output image created by real time 3D graphics engines (such as game engines). In this example the implementation is based on one of the most robust and powerful game engines, Unreal Engine 5. This example will illustrate minor benefits which are done by the algorithm tailored for specific project's needs, which are useful in case of not very high total polygon amount per scene. The visual output might not be the desired result in all cases, but it might be applied in many scenarios.

За основу даного методу було взято метод рендерінгу плиткової відкладеної тіні та використання тіньових карт. Цей метод включає 5 етапів виконання, а саме підготовку, заповнення, зважування, складання та перенесення.

Під час підготовки на процесорі обчислюється новий вид камери, зріз якої обмежений найменшим полем зору між межею персонажа та межею світла. Отримана додаткова тіньова карта займає 2048x2048 пікселів, але кожна тінь у результаті зменшує роздільну здатність на основі оцінки покриття екрану світлом. Також очищуються попередні тіньові карти на відеокарті на початку кожної відкладеної тіні.

У заповненні алгоритм уникає рендерінгу меша персонажів більше аніж один раз, перенаправляючи виклики рисування від стандартного буферу до додаткової тіньової карти. Після цього зважується ієрархічний буфер глибини та генерується карта тіней з міпмапою, зберігаючи найближчу відстань до світла у кожних 2 × 2 пікселях.

На етапі складання на процесорі буде створено об'єм тіні персонажа, простягаючи промені від джерела світла до позицій кожного обмеження, доки не буде досягнуто радіус світла. Після проекції в екранних координатах алгоритм знаходить мінімальну та максимальну позиції для запуску обчислювальних робочих одиниць, обмежених задіяними тіньовими плитками. У обчислювальному шейдері кожна робоча одиниця шукає відповідний світловий індекс у своєму записі буфера фінального списку світла. Якщо його не знайдено або позначено як повністю затінений, то алгоритм припиняє роботу. В іншому випадку оцінюється видимість світла кожного пікселя, і якщо вся плитка тіні повністю

Рисунок Г.3 – Матеріал тез, сторінка 1

освітлена, алгоритм зупиняється на цьому. Коли виявляється часткова тінь, гарантовано, що виділено слот видимості та збережено нову зменшену видимість світла.

У перенесенні кожна високоякісна тінь обробляється окремо з повторним використанням спільної цілі візуалізації результату алгоритму. Це має перевагу в тому, що вартість обчислень є єдиним обмежуючим фактором, оскільки накладні витрати пам'яті фіксовані. Однак вміст тіньової карти відкидається між кожною парою символ/світло, що робить його недоступним для прямого рендерінга. Оскільки персонажі з високоякісними тінями не малюються в них, кожен результат алгоритму переноситься назад у пов'язану стандартну тіньову карту. Це досягається шляхом проектування квадрата у тіньовій карті у ближній площині в стандартні координати тіньової карти світла. Один стандартний піксель тіньової карти містить багато пікселів результату алгоритму, і зчитування кожного з них було б непомірно дорогим. Ця проблема з високою пропускною здатністю вирішується за допомогою версії алгоритму із міпмапою, де кожен піксель вибирає рівень міпмапи, якому потрібні дві вибірки для покриття найменшого розміру XY області. Потім виконується повторне зчитування значень результату алгоритму з міпмапою, доки не буде покрито всю область. Наприклад, якщо піксель проектується в область розміром 12 x 34 пікселі у результаті алгоритму, то буде обрано 3 рівень міпмапи.

На таблиці 1 відображено результати оптимізації у порівнянні з тінями, вбудованими в UE5. Такі тіні потребують лише на 55% часу більше, ніж без тіней і на 7% менше часу за Епіс тіні, при цьому не втрачають у їх якості у випадку проектів з малою кількістю полігонів на одній сцені.

Таблиця 1 – Часу рендерінгу одного кадру алгоритму та без нього

	Час (мс)	Різниця
UE5 без тіней	3.29	
UE5 з Епіс тінями	5.48	+66%
UE5 з тінями алгоритму	5.12	+55%

У результаті даної роботи було створено оптимізаційний алгоритм для рендерінгу кадрів у 3D додатках реального часу і протиставлено результат ігровому рушію UE5.

Список використаних джерел:

1. Bart W. Cull that cone! Improved cone/spotlight visibility tests for tiled and clustered lighting. Apr. 2017. url: https://bartwronski.com/2017/04/13/cull-that-cone (дата звернення: 23.02.2024).

2. Nathan R. Depth Precision Visualized. NVidia. July 2015. url: https://developer.nvidia.com/content/depth-precision-visualized (дата звернення: 19.02.2024).

Рисунок Г.4 – Матеріал тез, сторінка 2

ДОДАТОК Ґ

Тези доповіді для науково-практичної інтернет-виставки

10. Ігровий програмний застосунок у жанрі Roguelite RPG. Bounty Chase

Автори: ***Донець Дмитро Сергійович,*** ст. гр. ПЗПІ-20-2, ***Вожова Маргарита Володимирівна,*** ст. гр. ПЗПІ-20-5, ХНУРЕ.

Науковий керівник: Новіков Юрій Сергійович, старший викладач. каф. ПІ, ХНУРЕ.

Ігровий програмний застосунок у жанрі Roguelite RPG “Bounty Chase” створений для проведення вільного часу.

Основа мета гри – проходження якомога більшої кількості хвиль ворогів, між якими гравець може покращувати свої атрибути та здібності, або відкривати нові для нього. Окрім цього гравець після поразки повертається сильнішим, маючи сильніші здібності і може проходити гру знову за бажанням. Вороги посилюються з рівнем. Декілька гравців можуть проходити разом один і той самий рівень одночасно.

Переваги розробки: кросплатформеність, простий та зрозумілий інтерфейс, легке впровадження дизайнерами нових значень за допомогою таблиць, підтримка мультиплеєрної гри.

Рисунок Ґ.1 – Опис роботи для презентації

ДОДАТОК Д

Приклад програмного коду

```
void UBountyAbilitySystemComponent::AbilityActorInfoSet()

{

	OnGameplayEffectAppliedDelegateToSelf.AddUObject(this,
&UBountyAbilitySystemComponent::ClientEffectApplied);

}


void UBountyAbilitySystemComponent::AddCharacterAbilities(const
TArray<TSubclassOf<UGameplayAbility>>& StartupAbilities)

{

	for (const TSubclassOf<UGameplayAbility> AbilityClass : StartupAbilities)

	{

		FGameplayAbilitySpec AbilitySpec = FGameplayAbilitySpec(AbilityClass, 1);

		if (const UBountyGameplayAbility* BountyAbility =
Cast<UBountyGameplayAbility>(AbilitySpec.Ability))

		{

			AbilitySpec.DynamicAbilityTags.AddTag(BountyAbility->StartupInputTag);


	AbilitySpec.DynamicAbilityTags.AddTag(FBountyGameplayTags::Get().Abilities_Status_Equ
ipped);

			GiveAbility(AbilitySpec);

		}

	}

	bStartupAbilitiesGiven = true;

	AbilitiesGivenDelegate.Broadcast();

}


void UBountyAbilitySystemComponent::AddCharacterPassiveAbilities(

	const TArray<TSubclassOf<UGameplayAbility>>& StartupPassiveAbilities)

{

	for (const TSubclassOf<UGameplayAbility> AbilityClass : StartupPassiveAbilities)

	{

		FGameplayAbilitySpec AbilitySpec = FGameplayAbilitySpec(AbilityClass, 1);

		GiveAbilityAndActivateOnce(AbilitySpec);

	}

}


void UBountyAbilitySystemComponent::AbilityInputTagHeld(const FGameplayTag& InputTag)
```

```
{
	if (!InputTag.IsValid()) return;

	for (FGameplayAbilitySpec& AbilitySpec : GetActivatableAbilities())
	{
		if (AbilitySpec.DynamicAbilityTags.HasTagExact(InputTag))
		{
			AbilitySpecInputPressed(AbilitySpec);
			if (!AbilitySpec.IsActive())
			{
				TryActivateAbility(AbilitySpec.Handle);
			}
		}
	}
}

void UBountyAbilitySystemComponent::AbilityInputTagReleased(const FGameplayTag& InputTag)
{
	if (!InputTag.IsValid()) return;

	for (FGameplayAbilitySpec& AbilitySpec : GetActivatableAbilities())
	{
		if (AbilitySpec.DynamicAbilityTags.HasTagExact(InputTag))
		{
			AbilitySpecInputReleased(AbilitySpec);
		}
	}
}

void UBountyAbilitySystemComponent::ForEachAbility(const FForEachAbility& Delegate)
{
	FScopedAbilityListLock ActiveScopeLock(*this);
	for (const FGameplayAbilitySpec& AbilitySpec : GetActivatableAbilities())
	{
		if (!Delegate.ExecuteIfBound(AbilitySpec))
		{
			UE_LOG(LogBounty, Error, TEXT("Failed to execute delegate in %hs"),
__FUNCTION__);
		}
```

```
	}
}


FGameplayTag UBountyAbilitySystemComponent::GetAbilityTagFromSpec(const
FGameplayAbilitySpec& AbilitySpec)
{
	if (AbilitySpec.Ability)
	{
		for (FGameplayTag Tag : AbilitySpec.Ability.Get()->AbilityTags)
		{
			if
(Tag.MatchesTag(FGameplayTag::RequestGameplayTag(FName("Abilities"))))
			{
				return Tag;
			}
		}
	}
	return FGameplayTag();
}


FGameplayTag UBountyAbilitySystemComponent::GetInputTagFromSpec(const FGameplayAbilitySpec&
AbilitySpec)
{
	for (FGameplayTag Tag : AbilitySpec.DynamicAbilityTags)
	{
		if (Tag.MatchesTag(FGameplayTag::RequestGameplayTag(FName("InputTag"))))
		{
			return Tag;
		}
	}
	return FGameplayTag();
}


FGameplayTag UBountyAbilitySystemComponent::GetStatusFromSpec(const FGameplayAbilitySpec&
AbilitySpec)
{
	for (FGameplayTag StatusTag : AbilitySpec.DynamicAbilityTags)
	{
		if
(StatusTag.MatchesTag(FGameplayTag::RequestGameplayTag(FName("Abilities.Status"))))
```

```
			{
				return StatusTag;
			}
		}
		return FGameplayTag();
}

FGameplayTag UBountyAbilitySystemComponent::GetStatusFromAbilityTag(const FGameplayTag&
AbilityTag)
{
		if (const FGameplayAbilitySpec* Spec = GetSpecFromAbilityTag(AbilityTag))
		{
			return GetStatusFromSpec(*Spec);
		}
		return FGameplayTag();
}

FGameplayTag UBountyAbilitySystemComponent::GetInputTagFromAbilityTag(const FGameplayTag&
AbilityTag)
{
		if (const FGameplayAbilitySpec* Spec = GetSpecFromAbilityTag(AbilityTag))
		{
			return GetInputTagFromSpec(*Spec);
		}
		return FGameplayTag();
}

FGameplayAbilitySpec* UBountyAbilitySystemComponent::GetSpecFromAbilityTag(const
FGameplayTag& AbilityTag)
{
		FScopedAbilityListLock ActiveScopeLoc(*this);
		for (FGameplayAbilitySpec& AbilitySpec : GetActivatableAbilities())
		{
			for (FGameplayTag Tag : AbilitySpec.Ability.Get()->AbilityTags)
			{
				if (Tag.MatchesTag(AbilityTag))
				{
					return &AbilitySpec;
				}
```

```
			}
		}
		return nullptr;
}

void UBountyAbilitySystemComponent::UpdateAbilityStatuses(int32 Level)
{
		UAbilityInfo* AbilityInfo = UBountyAbilitySystemLibrary::GetAbilityInfo(GetAvatarActor());
		for (const FBountyAbilityInfo& Info : AbilityInfo->AbilityInformation)
		{
			if (!Info.AbilityTag.IsValid()) continue;
			if (Level < Info.LevelRequirement) continue;
			if (GetSpecFromAbilityTag(Info.AbilityTag) == nullptr)
			{
				FGameplayAbilitySpec AbilitySpec = FGameplayAbilitySpec(Info.Ability, 1);
		AbilitySpec.DynamicAbilityTags.AddTag(FBountyGameplayTags::Get().Abilities_Status_Eligible);
				GiveAbility(AbilitySpec);
				MarkAbilitySpecDirty(AbilitySpec);
		ClientUpdateAbilityStatus(Info.AbilityTag,FBountyGameplayTags::Get().Abilities_Status_Eligible, 1);
			}
		}
}

void UBountyAbilitySystemComponent::ServerSpendSpellPoint_Implementation(const FGameplayTag& AbilityTag)
{
		if (FGameplayAbilitySpec* AbilitySpec = GetSpecFromAbilityTag(AbilityTag))
		{
			if (GetAvatarActor()->Implements<UPlayerInterface>())
			{
				IPlayerInterface::Execute_AddToSpellPoints(GetAvatarActor(), -1);
			}

			const FBountyGameplayTags GameplayTags = FBountyGameplayTags::Get();
			FGameplayTag Status = GetStatusFromSpec(*AbilitySpec);
```

```
            if (Status.MatchesTagExact(GameplayTags.Abilities_Status_Eligible))

            {

                  AbilitySpec-
>DynamicAbilityTags.RemoveTag(GameplayTags.Abilities_Status_Eligible);

                  AbilitySpec-
>DynamicAbilityTags.AddTag(GameplayTags.Abilities_Status_Unlocked);

                  Status = GameplayTags.Abilities_Status_Unlocked;

            }

            else if (Status.MatchesTagExact(GameplayTags.Abilities_Status_Equipped) ||
Status.MatchesTagExact(GameplayTags.Abilities_Status_Unlocked))

            {

                  AbilitySpec->Level += 1;

            }

            ClientUpdateAbilityStatus(AbilityTag, Status, AbilitySpec->Level);

            MarkAbilitySpecDirty(*AbilitySpec);

      }

}
```