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 в жанрі Shoot'em-up. Алгоритми процедурної генерації рівнів, робота з освітленням, механіки взаємодії з оточенням
(тема)

Виконав:
студент 4 курсу, групи ПЗПІ-20-7

Попов Д. М.
(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 ас. кафедри ПІ Матвєєв Д. І.
(посада, прізвище, ініціали)

Допускається до захисту
Зав. кафедри _____ З. В. Дудар
(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
Тип програми Освітньо-професійна
Освітня програма Програмна Інженерія
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Попову Дмитру Максимовичу
(прізвище, ім’я, по батькові)

1. Тема роботи Ігровий програмний застосунок в жанрі Shoot'em-up. Алгоритми процедурної генерації рівнів, робота з освітленням, механіки взаємодії з оточенням.

Затверджена наказом по університету від 20.05.2024р. № 471 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 10.06.2024

3. Вихідні дані до роботи *Розробити систему випадкової генерації ландшафту та програмні інструменти для автоматичного створення езкемплярів ландшафту та декорацій. Також сторити самі екземпляри ландшафту та декорацій для генерації. Реалізувати різні механіки генерації. Реалізацію системи виконати на мові програмування C# та ігровому двигуні Unity3D*

4. Перелік питань, що потрібно опрацювати в роботі

Вступ, аналіз предметної галузі, формування вимог до програмної системи, архітектура та проектування програмного забезпечення, опис прийнятих програмних рішень, тестування розробленого програмного забезпечення, висновки, додатки.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі	15.04.2024	*виконано*
2	Створення специфікації ПЗ	22.04.2024	*виконано*
3	Проектування ПЗ	29.04.2024	*виконано*
4	Розробка ПЗ	06.04.2024	*виконано*
5	Тестування ПЗ	13.05.2024	*виконано*
6	Оформлення пояснювальної записки	20.05.2024	*виконано*
7	Підготовка презентації та доповіді	27.05.2024	*виконано*
8	Попередній захист	06.05.2024	*виконано*
9	Нормоконтроль, рецензування	06.05.2024	*виконано*
10	Здача роботи у електронний архів	07.06.2024	*виконано*
11	Допуск до захисту у зав. кафедри	09.06.2024	*виконано*

Дата видачі завдання 8 квітня 2024р.

Студент (ка) _____ Попов Д. М.
(підпис)

Керівник роботи _____ ас. кафедри ПІ Матвєєв Д. І.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка до кваліфікаційної роботи бакалавра: 83 с., 57 рис., 10 джерел, 8 таблиць.

АЛГОРИТМ, ЛАНДШАФТ, ДЕКОРАЦІЇ, ІГРОВИЙ ПРОГРАМНИЙ ЗАСТОСУНОК, ГЕНЕРАЦІЯ, C#, UNITY3D, ВИПАДКОВА ГЕНЕРАЦІЯ ЛАНДШАФТУ, SHOOT'EM UP.

Об'єктом даної роботи є розробка системи генерації ландшафту та декорацій для ігрового програмного застосунку у жанрі shoot`em up. Робота включає створення програмних інструментів для автоматичного створення екземплярів ландшафту та декорацій, а також реалізацію механік генерації.

Метою роботи є створення системи, яка дозволить розробникам швидко та ефективно генерувати різноманітні ландшафти та декорації, забезпечуючи унікальність кожної ігрової сесії. Система повинна бути гнучкою, щоб дозволити легке внесення змін та доповнень до ігрового світу.

Методика розробки базується на використанні ігрового двигуна Unity3D та мови програмування C# для створення стабільної та оптимізованої системи генерації. Для візуального представлення ландшафту та декорацій використовуються засоби графічного дизайну, такі як Adobe Photoshop та Illustrator.

У результаті роботи було розроблено комплексний підхід до генерації ландшафту та декорацій, що включає алгоритми випадкової генерації, систему шаблонів для створення екземплярів та інструменти для їх редагування. Створено прототип системи, який демонструє можливості генерації та інтеграції з ігровим світом.

ALGORITHM, LANDSCAPE, DECORATIONS, GAME SOFTWARE APPLICATION, GENERATION, C#, UNITY3D, RANDOM LANDSCAPE GENERATION, SHOOT'EM UP.

The object of this work is the development of a landscape and decorations generation system for a game software application in the shoot'em up genre. The work includes the creation of software tools for the automatic generation of landscape and decoration instances, as well as the implementation of generation mechanics.

The purpose of the work is to create a system that allows developers to quickly and efficiently generate various landscapes and decorations, ensuring the uniqueness of each gaming session. The system should be flexible to allow easy changes and additions to the game world.

The development methodology is based on the use of the Unity3D game engine and the C# programming language to create a stable and optimized generation system. Graphic design tools such as Adobe Photoshop and Illustrator are used for the visual representation of the landscape and decorations.

As a result of the work, a comprehensive approach to the generation of landscapes and decorations was developed, which includes random generation algorithms, a template system for creating instances, and tools for their editing. A prototype system has been created that demonstrates the possibilities of generation and integration with the game world.

Я, Попов Дмитро Максимович, студент гр. ПЗПІ-20-7, здобувач вищої освіти на першому (бакалаврському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему « Ігровий програмний застосунок в жанрі Shoot'em-up. Алгоритми процедурної генерації рівнів, робота з освітленням, механіки взаємодії з оточенням», 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

ПЕРЕЛІК СКОРОЧЕНЬ

PC – Personal Computer

ОС – Операційна Система

ВСТУП

В епоху цифрових технологій, коли віртуальний світ стає все більш реалістичним та захоплюючим, жанр комп’ютерних ігор shoot 'em up [1] продовжує зберігати свою актуальність, пропонуючи гравцям неперевершені відчуття динаміки та адреналіну. Ці ігри, які вимагають від гравців не лише швидкості реакції, але й стратегічного планування та тактичної обачності, є втіленням справжнього виклику для любителів ігрової індустрії.

З розвитком технологій та зростанням можливостей програмування, жанр shoot 'em up перетворився з простих аркадних ігор у складні інтерактивні світи, де кожен рівень, кожен ворог, кожна зброя має свою унікальну історію та характеристики. Це не просто ігри; це цілісні всесвіти, які занурюють гравця у незабутні пригоди, де кожен вибір має значення, а кожен рух може визначити результат битви.

Сучасні технології дозволяють розробникам створювати ігри, які не тільки виглядають красиво, але й відчуваються реалістично. Від фізики руху до візуальних ефектів – кожен аспект ігри ретельно продуманий, щоб забезпечити найкращий ігровий досвід. Ігри жанру shoot 'em up вимагають від гравців не тільки швидкості, але й точності, стратегії та, іноді, командної взаємодії.

Історія жанру shoot 'em up – це історія інновацій та креативності. Від перших текстових ігор до сучасних віртуальних реальностей, цей жанр завжди був на передньому краї ігрової індустрії, пропонуючи нові та захоплюючі способи гри. Це історія про те, як прості концепції можуть перетворитися на складні ігрові системи, які захоплюють мільйони гравців по всьому світу.

З самого свого зародження в ранніх 60-х роках минулого століття, коли перші ігри, такі як “Space Invaders” [2], встановили основи для майбутніх поколінь ігрових розробок, жанр shoot 'em up пройшов довгий шлях розвитку. Від простих двовимірних сценаріїв до складних тривимірних віртуальних світів, цей жанр не перестає дивувати та надихати розробників на створення нових ігрових шедеврів.

У центрі уваги даної кваліфікаційної роботи знаходиться розробка системи генерації ландшафту та декорацій для ігрового застосунку у жанрі shoot 'em up.

Ландшафт та декорації в іграх цього жанру відіграють не лише естетичну роль, але й є невід’ємною частиною геймплею, впливаючи на тактику та стратегію гравця. Вони створюють унікальний ігровий простір, який з кожною новою сесією пропонує гравцям нові виклики та відкриває перед ними незвідані горизонти.

Метою даної роботи є створення інноваційної та адаптивної системи, яка забезпечить розробникам можливість швидкого та ефективного створення різноманітних ландшафтів та декорацій, підвищуючи при цьому унікальність кожної ігрової сесії. Система повинна бути легко інтегрована у існуючий ігровий двигун, забезпечуючи при цьому можливість легкого внесення змін та доповнень до ігрового світу.

У процесі роботи буде проведено глибокий аналіз існуючих рішень у цій області, визначено ключові вимоги до системи, розроблено архітектуру та алгоритми генерації, створено прототип системи та перевірено її на практиці. Особлива увага буде приділена оптимізації та забезпеченню високої продуктивності системи, щоб вона могла працювати в реальному часі, не впливаючи на продуктивність гри.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної галузі

Основою успіху відеогри є її здатність викликати інтерес та задоволення у гравців, які вирішують її придбати. Важливо, щоб гра не тільки привертала увагу своїм зовнішнім виглядом та обіцянками на етапі маркетингу, але й продовжувала радувати гравця під час самої гри. Розчарування після покупки може призвести до негативних відгуків та шкоди репутації продукту. З іншого боку, якщо гра виправдовує очікування, це сприяє позитивному відгуку та може стимулювати розробку доповнень або продовжень. Таким чином, ключовим є глибоке розуміння того, які ігрові елементи та характеристики найбільше приваблюють гравців у вибраному жанрі. Аналіз, отже, повинен бути зосереджений на ігрових механіках та особливостях, які є фундаментальними для жанру, а також на тих, що використовуються в аналогічних успішних іграх.

У сучасному світі відеоігор, системи генерації ландшафту відіграють важливу роль у створенні динамічного та занурюючого ігрового середовища. Вони не тільки забезпечують унікальний досвід для кожного гравця, але й значно розширюють можливості повторного проходження гри, що є критично важливим для жанру shoot 'em up. Генеративні алгоритми дозволяють створювати ландшафти, які вражають своєю неповторністю та красою, водночас зберігаючи ігровий світ свіжим та захоплюючим. Це особливо важливо для жанрів, де світ гри є ключовим елементом досвіду, як у випадку з shoot 'em up, де середовище може стати як союзником, так і ворогом. Системи генерації ландшафту також сприяють розширенню ігрового простору без необхідності ручного моделювання кожної деталі, що економить час та ресурси розробників. Вони відкривають двері для безкінечних варіацій ігрових рівнів, що підтримує інтерес гравців на високому рівні. Завдяки цьому, ігри з системами генерації ландшафту часто мають високу вартість повторного проходження, оскільки кожне нове гравецьке сесія пропонує нові враження та виклики.

Terraria [3] та Minecraft [4] є яскравими прикладами ігор, де генерація світу відбувається випадковим чином, пропонуючи гравцям неповторні ландшафти та

виклики з кожним новим створенням світу. Ці ігри демонструють, як система генерації може впливати на ігровий процес, змушуючи гравців адаптуватися та використовувати різні стратегії. Вони створюють глибокий ігровий світ, де кожен камінь, кожна рослина, і кожна крапля води можуть мати значення для гравця (див. рисунок 1.1). Це не тільки додає глибини геймплею, але й спонукає гравців до дослідження та експериментування. У таких іграх, кожен новий світ є загадкою, яку гравці мають розгадати, використовуючи свої навички та інтуїцію. Це створює відчуття пригоди та відкриття, яке є важливим для залучення та утримання гравців. Такий підхід до генерації світу також сприяє створенню сильної спільноти навколо гри, оскільки гравці обмінюються своїми відкриттями та історіями.

Рисунок 1.1 – Генерація світу в Minecraft (зроблено власноруч)

У контексті shoot 'em up, де швидкість та реакція є ключовими, генерація ландшафту може додати додатковий рівень складності та непередбачуваності. Такі ігри, як Starbound [5] або No Man’s Sky [6], де експлорація є основною частиною досвіду, показують, як різноманітність ландшафтів може збагатити геймплей (див. рисунок 1.2).

Рисунок 1.2 – Експлорація планети в No Man’s Sky (зроблено власноруч)

Ці ігри вводять концепцію безкінечного простору для дослідження, де кожна планета або зона може мати свої унікальні властивості та виклики. Гравці можуть зіткнутися з різними типами теренів, від пустель до лісів, кожен з яких вимагає від гравців особливого підходу (див. рисунок 1.3 та 1.4). Це додає глибини стратегічному плануванню та робить кожну битву унікальною. Використання генерації ландшафту в shoot 'em up іграх може створити нові можливості для геймдизайну, дозволяючи розробникам відходити від традиційних лінійних рівнів та пропонувати гравцям більш відкритий та інтерактивний досвід. Такий підхід може збільшити вартість повторного проходження гри, оскільки гравці будуть мати бажання повертатися до гри, щоб дослідити нові генеровані світи та випробувати нові стратегії.

Рисунок 1.3 – Пустельна планета в Starbound (зроблено власноруч)

Процедурна генерація дозволяє створювати величезні, унікальні світи з високим рівнем варіативності, як у No Man’s Sky чи Starbound, зменшуючи при цьому необхідність ручної роботи.

Рисунок 1.4 – Газовий Гігант в Starbound (зроблено власноруч)

Фундаментальним аспектом успіху відеоігор є їх здатність викликати зацікавленість та задоволення у гравців, які вирішують їх придбати. Це вимагає від розробників не лише створення привабливого зовнішнього вигляду та ефективної

маркетингової стратегії, але й забезпечення того, щоб гра продовжувала радувати гравця під час самої гри. Розчарування після покупки може призвести до негативних відгуків та шкоди репутації продукту, тоді як позитивний ігровий досвід може сприяти створенню доповнень або продовжень. Таким чином, ключовим є глибоке розуміння того, які ігрові елементи та характеристики найбільше приваблюють гравців у вибраному жанрі. Аналіз повинен бути зосереджений на ігрових механіках та особливостях, які є фундаментальними для жанру, а також на тих, що використовуються в аналогічних успішних іграх.

Системи генерації ландшафту відіграють важливу роль у створенні динамічного та занурюючого ігрового середовища. Вони забезпечують унікальний досвід для кожного гравця та розширюють можливості повторного проходження гри, що є критично важливим для жанру shoot ‘em up. Генеративні алгоритми створюють ландшафти, які вражають своєю неповторністю та красою, зберігаючи ігровий світ свіжим та захоплюючим. Це особливо важливо для жанрів, де світ гри є ключовим елементом досвіду, як у випадку з shoot ‘em up, де середовище може стати як союзником, так і ворогом. Системи генерації ландшафту сприяють розширенню ігрового простору без необхідності ручного моделювання кожної деталі, що економить час та ресурси розробників. Вони відкривають двері для безкінечних варіацій ігрових рівнів, підтримуючи інтерес гравців на високому рівні. Ігри з системами генерації ландшафту часто мають високу вартість повторного проходження, оскільки кожна нова гравецька сесія пропонує нові враження та виклики.

Адаптивність цих систем також дозволяє гравцям впливати на світ, змінюючи ландшафт за допомогою своїх дій, що додає глибини ігровому процесу. Адаптивність систем генерації ландшафту також може бути інтегрована з іншими ігровими системами, такими як погода, час доби, або навіть поведінка NPC, що робить кожну гравецьку сесію ще більш унікальною та непередбачуваною. Це створює додатковий рівень інтерактивності, коли гравці можуть бачити безпосередній вплив своїх дій на світ, що навколо них. Така динаміка змушує гравців бути більш уважними та реактивними до змін у середовищі, що може

вплинути на їхні стратегії виживання та прогресування в грі. В кінцевому підсумку, це збільшує вартість повторного проходження та залученість гравців, оскільки кожна сесія пропонує новий набір викликів та можливостей для дослідження. Дизайн-документ розроблюваної гри наведено у додатку В.

1.2 Виявлення проблем та актуалізація рішень

Після аналізу предметної галузі стає зрозуміло, що для досягнення успіху в ігровій індустрії, гра повинна мати унікальні особливості та сильні сторони, які відповідають вимогам обраного жанру. У жанрі shoot 'em up, де акцент робиться на швидкість дії та реакцію, важливо зосередитися на таких аспектах геймплею, як веселий та динамічний геймплей, вражаюча генерація місцевості, атмосферні декорації, унікальна та різноманітна зброя, велика кількість різних ворогів, прогресія розвитку персонажа, складний, але збалансований геймплей, та вражаюча візуальна складова проекту.

Однією з ключових проблем, яку необхідно вирішити, є динамічна генерація ландшафту та місцевості. Це вимагає розробки алгоритмів, які можуть швидко та ефективно створювати переконливі та різноманітні сцени, що є важливим для підтримки інтересу та залученості гравців. Процес повинен бути оптимізований таким чином, щоб забезпечити високу продуктивність навіть на обмеженому апаратному забезпеченні, що є викликом для розробників. Для досягнення цього, розробники використовують різні складні технології, що дозволяють імітувати природні форми ландшафту. Також важливою є інтеграція штучного інтелекту для адаптації генерованих світів під поведінку та переваги гравців, забезпечуючи унікальний ігровий досвід.

Вимоги до продуктивності обладнання, висока деталізація та динаміка ігрового світу можуть вимагати значних обчислювальних ресурсів, особливо від графічного процесора (GPU) та центрального процесора (CPU). Це може стати проблемою для гравців зі слабшими системами [7]. Тому, оптимізація генерації ландшафту є критичною для забезпечення доступності гри на широкому спектрі обладнання. Розробники повинні знайти способи зменшення навантаження на

систему, не жертвуючи якістю візуальних ефектів. Це може включати в себе алгоритми кешування, розумне використання LOD (Level of Detail), та інші техніки, які дозволяють зменшити кількість полігонів без втрати візуальної якості.

Шум Перліна [8] є одним із популярних методів для процедурної генерації ландшафту, який дозволяє створювати природні, реалістичні форми (див. рисунок 1.5). Використання таких алгоритмів дозволяє розробникам створювати детальні та візуально привабливі ландшафти без необхідності вручну моделювати кожен елемент сцени. Однак, це також створює виклики, пов’язані з інтеграцією цих ландшафтів у ігрову механіку, так щоб вони не тільки виглядали добре, але й сприяли геймплею.

Рисунок 1.5 – Ландшафт у грі Minecraft, створений за допомогою Шуму Перліна (за даними [9])

Для вирішення цих проблем, розробники можуть використовувати різні підходи, такі як оптимізація алгоритмів для кращої продуктивності та адаптація генерованих ландшафтів до ігрових механік. Це може включати регулювання складності ландшафту для забезпечення плавності руху персонажів та інтеграцію елементів, які можуть впливати на стратегію гравців, наприклад, шляхом створення природних перешкод або зон для засідок.

Важливо також враховувати тенденції в ігровій індустрії та очікування

гравців, адаптуючи стиль та рівень деталізації ландшафту до сучасних стандартів. Це може включати використання воксельної графіки для створення унікального візуального стилю або інтеграцію елементів ретро-дизайну для залучення шанувальників класичних ігор [10].

Загалом, динамічна генерація ландшафту та місцевості вимагає глибокого розуміння як технічних аспектів, так і ігрового дизайну, щоб створити баланс між візуальною привабливістю, продуктивністю та геймплеєм. Це вимагає постійних експериментів та ітерацій, щоб знайти оптимальні рішення, які задовольнять як розробників, так і гравців.

Щоб продукт міг конкурувати на ринку, важливо створити власні особливі та цікаві поєднання механік, які дозволять продукту виділятися на фоні інших. Проект повинен мати щось особливе та своє, що зробить його помітним серед конкурентів. Це можуть бути не лише інноваційні механіки, але й нові поєднання вже відомих механік, або новий погляд на старі механіки, застосовані нестандартним чином.

1.3 Постановка задачі

Під час реалізації проекту потрібно реалізувати автоматичну систему випадкової генерації ландшафту та декорацій. Реалізація повинна враховувати усі найважливіші аспекти системи, наприклад, такі як: програмна реалізація системи, автоматичний вибір ландшафту для генерації, автоматичний вибір декорації для обраного ландшафту, плавний перехід між шматами ландшафту, визначення висоти для генерації декорацій у різних точках ландшафту.

Система також повинна враховувати особливості системи переміщення противників, гравця та мати шляхи взаємодії із противниками (ШІ противників повинен мати змогу коректної постанови шляху до гравця) та гравцем (гравець повинен мати змогу обирати куди саме він захоче йти, повинен мати змогу вийти з кожного замкненого кута та ями, тощо). Система повинна також взаємодіяти із навколишнім середовищем: створювати плавні переходи між об’єктами ландшафту, обирати підходящі ландшафти для випадкової генерації, обирати підходящі декорації для певної локації. Система генерації ландшафту та декорацій

повинна враховувати обрану локацію та обмежувати випадкову генерацію ландшафту обраним типом ландшафтів. Для генерації повинні бути реалізовані різні параметри, щоб мати можливості створювати особливі локації, які матимуть свій ландшафт та декорації та не будуть не схожими між собою.

Для прискорення створення різних екземплярів ландшафту потрібно розробити програмний інструмент який повинен дозволяти швидко створювати ці екземпляри. Цей інструмент повинен надавати розробнику можливість створювати різні варіації екземплярів ландшафту по прикладу, змінюючи параметри екземплярів. Інструмент створення екземплярів ландшафту по прикладу має на меті прискорення процесу розробки екземплярів ландшафту шляхом можливості швидко завдавати та змінювати параметри, які повинні бути у будь якого екземпляру ландшафту будь-якої локації, при цьому залишивши можливість для модифікації параметрів генерації декорацій для створення особливих та не схожих між собою одна на одну локацій.

2 ПЕРЕЛІК ВИМОГ ДО СИСТЕМИ ГЕНЕРАЦІЇ ЛАНДШАФТУ

2.1 Постановка мети

Метою проекту є реалізація випадкової генерації ландшафту та декорацій та програмного інструменту для створення екземплярів ландшафту по прикладу. Система повинна бути виконана на ігровому рушії Unity3D на мові програмування C#. Вона повинна працювати на платформі PC на ОС Windows.

Програмний інструмент для генерації ландшафту повинен структурувати процес генерації ландшафту, використовуючи приклади для налаштування параметрів ландшафту. А також давати розробнику можливість створення особливих екземплярів ландшафту легким шляхом.

Система генерації ландшафту та декорацій повинна бути виконана враховуючи способи взаємодії з іншими системами гри (пересування противників, пересування гравця, система накопичення внутриігрових балів). Також система повинна задовільняти потреби гравця, створюючи неймовірні та, кожен раз, оригінальні розташування декорацій на карті. Гравець не повинен помічати переходів між різними екземплярами ландшафту. Також частина генерації повинна мати гарну візуальну складову та доречні атмосферні декорації.

2.2 Загальний опис

Система генерації ландшафту та декорацій для ігрового застосунку у жанрі shoot’em up є фундаментальною для створення захоплюючого ігрового середовища. Вона дозволяє гравцям зануритися в динамічний світ, де кожна деталь має значення для загального враження від гри. Система базується на комплексі алгоритмів та інструментів, які забезпечують процедурну генерацію унікальних ландшафтів, що не повторюються, та декорацій, які доповнюють ігровий процес.

Компоненти системи. Система включає в себе набір контролерів та інтерфейсів, які взаємодіють з іншими елементами ігрового застосунку. Контролери надають доступ до параметрів генерації, дозволяють налаштовувати рівень деталізації та складність ландшафту, а також відповідають за адаптацію генерованих об’єктів до поточних потреб геймплею.

Взаємодія з ігровими механіками. Система тісно інтегрована з ігровими механіками, забезпечуючи не тільки візуальну привабливість, але й впливаючи на геймплей. Наприклад, різноманітність ландшафту може впливати на тактику гравців, пропонуючи різні шляхи для проходження рівнів або створюючи природні перешкоди, які гравці повинні обійти.

Інструментарій для розробників. Розробники мають доступ до інструментарію, який дозволяє налаштовувати параметри генерації, вибирати типи декорацій та модифікувати ландшафт для досягнення бажаного ефекту. Це може включати редактори рівнів, конфігураційні файли та бібліотеки асетів, які спрощують процес створення ігрових світів.

Модульність та розширюваність. Система розроблена з урахуванням модульності, що дозволяє легко додавати нові функції та елементи без необхідності переписування існуючого коду. Конфігураційні файли розділені на вторинні модулі, кожен з яких відповідає за певний аспект генерації, що робить систему гнучкою та легкою для масштабування.

2.3 Загальні обмеження

Система генерації ландшафту та декорацій як й ігровий застосунок повинна бути реалізована на мові програмування C# та використовувати ігровий рушій Unity3D версії 2022.3.8f1. Система повинна працювати на ОС Windows 10/11.

3 АРХІТЕКТУРА ТА ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 UML проєктування ПЗ

Діаграма прецедентів є засобом візуалізації, що використовується для опису функціональних можливостей системи з точки зору користувача.

Для розроблюваного ігрового застосунку було розроблено діаграму прецедентів, що відповідають діям користувача у грі (див. рис. 3.1).

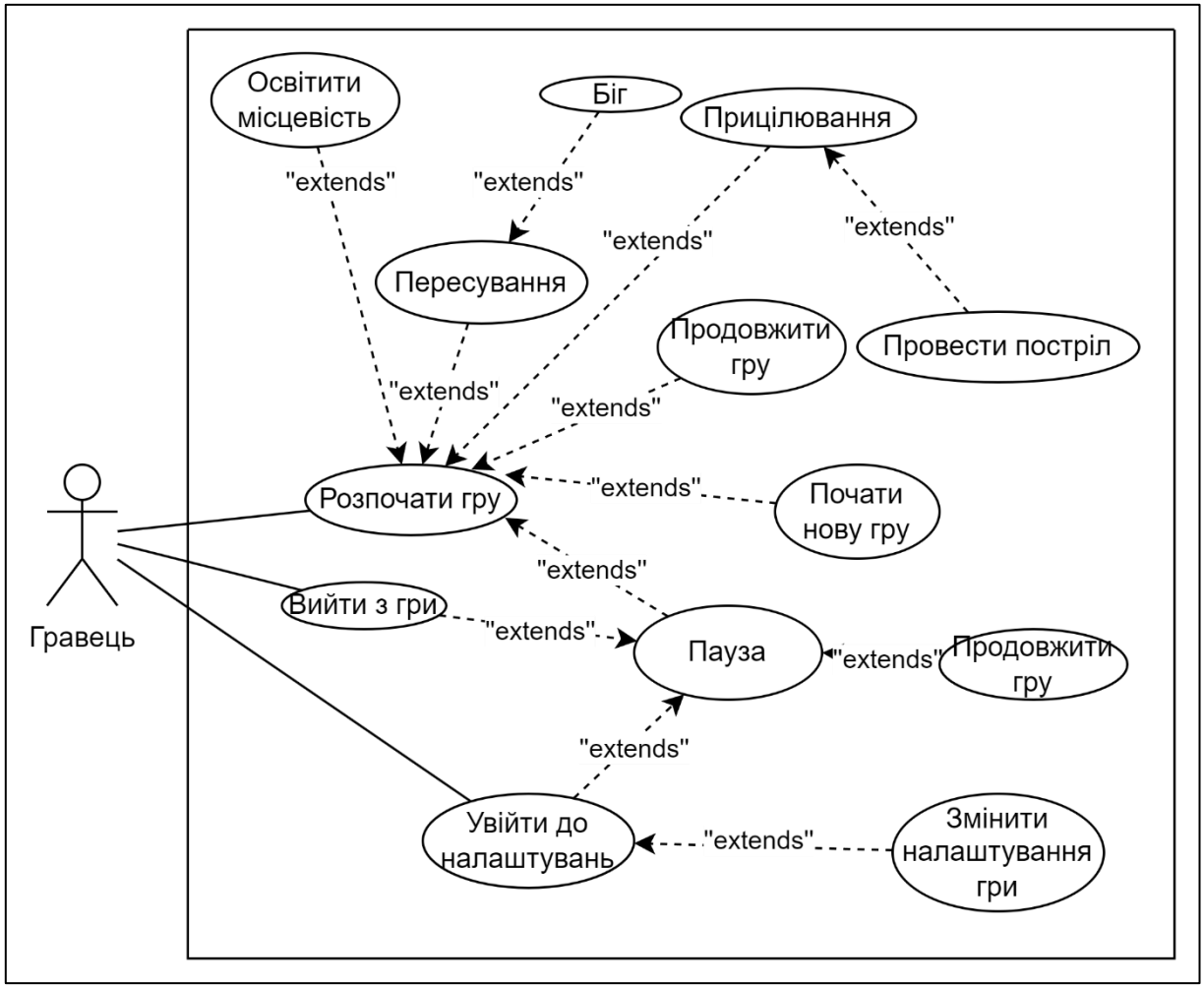


Рисунок 3.1 – Діаграма прецедентів (зроблено власноруч)

Ця діаграма відображає взаємодію між ігровим застосунком та гравцем через ідентифікацію основних дій, які можуть бути виконані в системі. Цей тип діаграми допомагає зрозуміти потреби користувачів та визначити основні функції системи з їхнього погляду.

3.2 Проектування архітектури

Діаграма класів для генерації ландшафту – це діаграма, яка використовується для того, що б продемонструвати зв’язки між класами програмного забезпечення, що відповідають за генерацію ландшафту (див. рисунок 3.2).

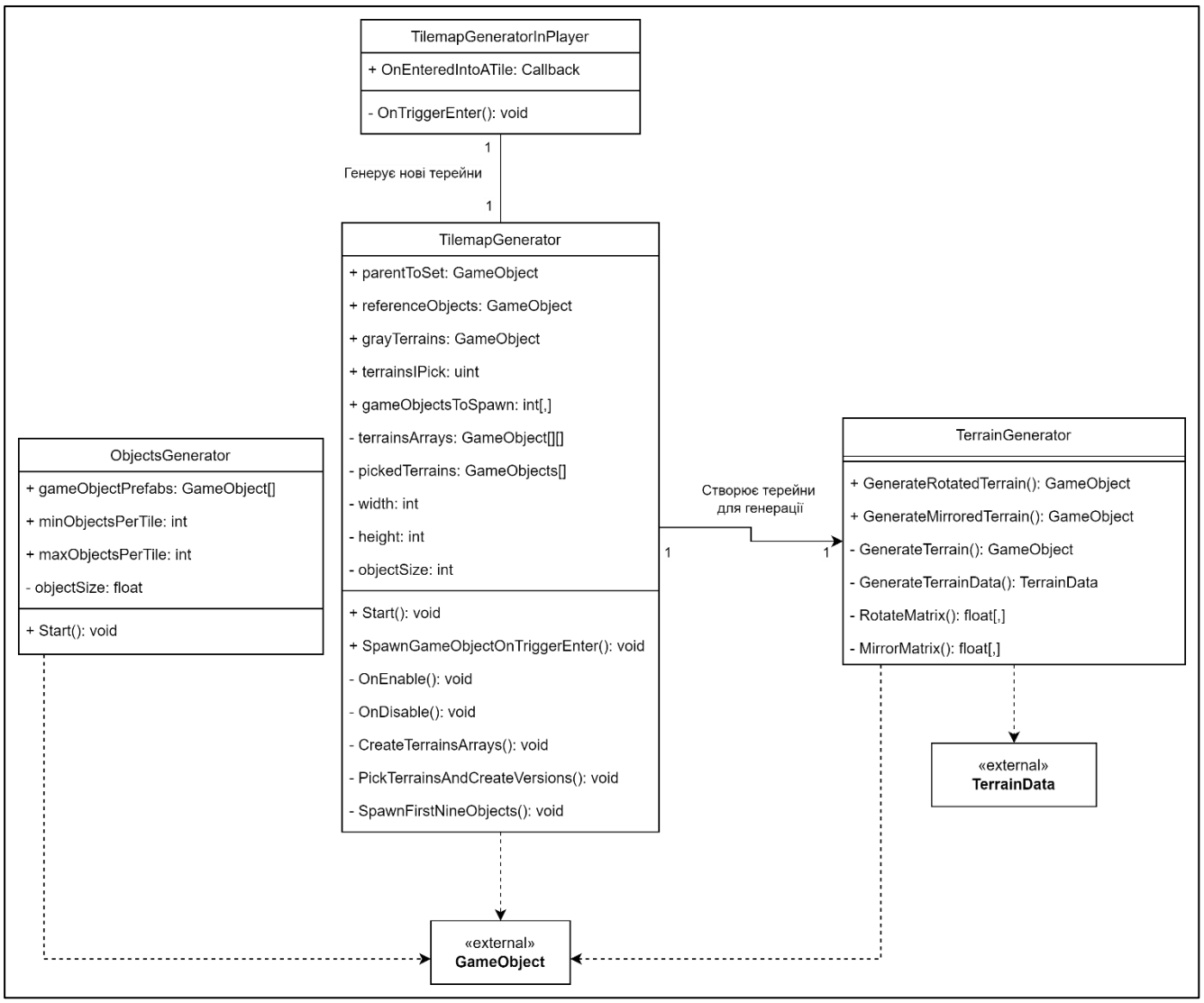


Рисунок 3.2 – Діаграма класів (зроблено власноруч)

Ця діаграма класів створена для демонстрації взаємодії класів генерації між собою та гравцем, а також демонструє, як генеруються декорації на ландшафті. Залучення класу PlayerController обумовлено тим, що гравець повинен рухатися за чіпляти трігери, які активують процес генерації.

3.3 Проектування алгоритму генерації ландшафтів та декорацій

Розроблено діаграму активності для демонстрації поведінки системи генерації та створення нового ландшафту (див. рисунок 3.3).

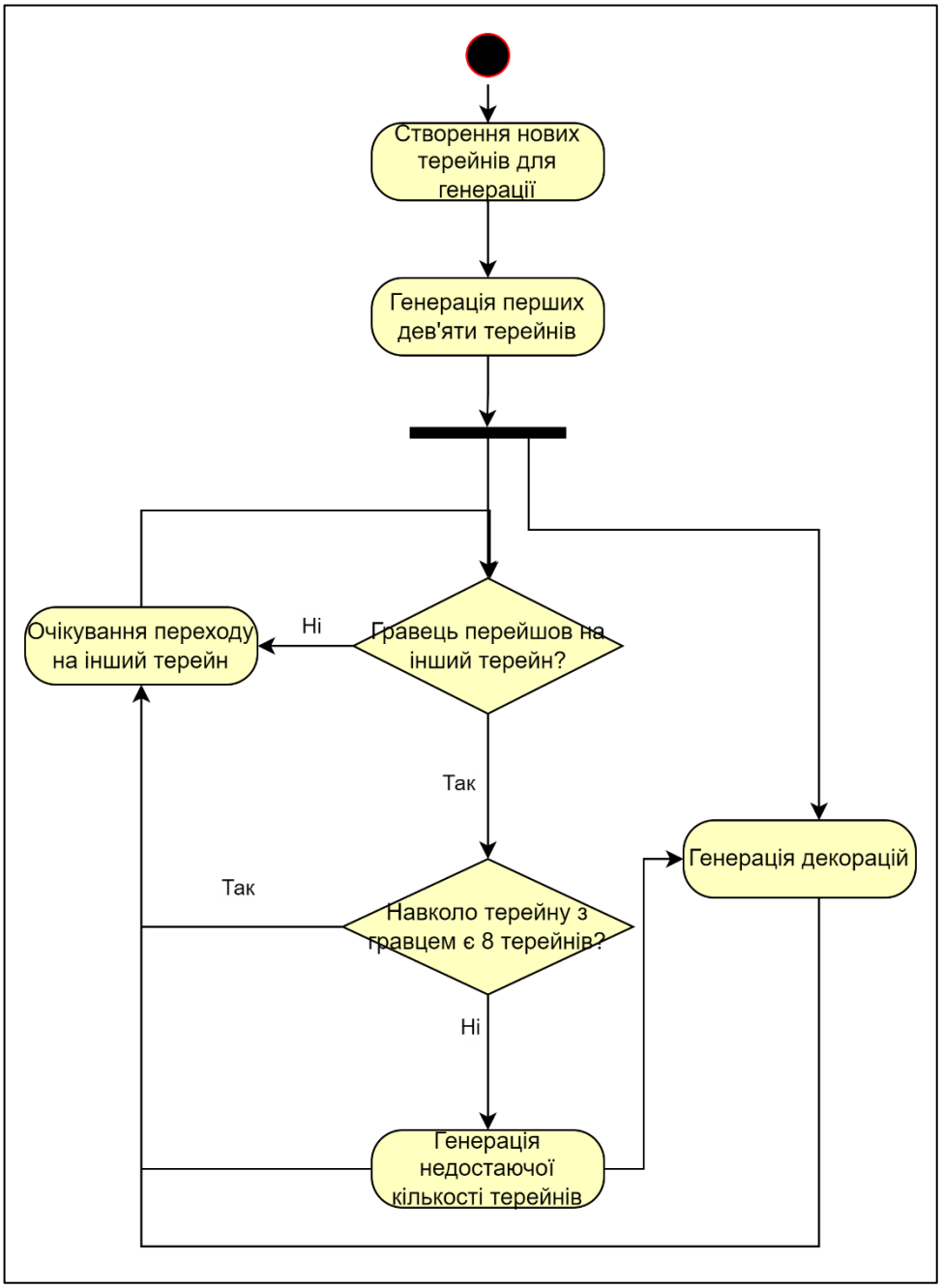


Рисунок 3.3 – Діаграма активності системи генерації ландшафту та декорацій (зроблено власноруч)

Ця діаграма показує, як саме відбувається створення та генерація ландшафту навколо гравця та на початку гри.

3.4 Проектування рівнів

Для гри потрібно розробити три локації, які відображають ті чи інші місця на планеті, чи за її межами.

Першою локацією я вирішив взяти поверхню Луни. Вона володіє красивими краєвидами з пагорбами та різної величини кратерами. Приклад такої локації зображено на рисунку 3.4.

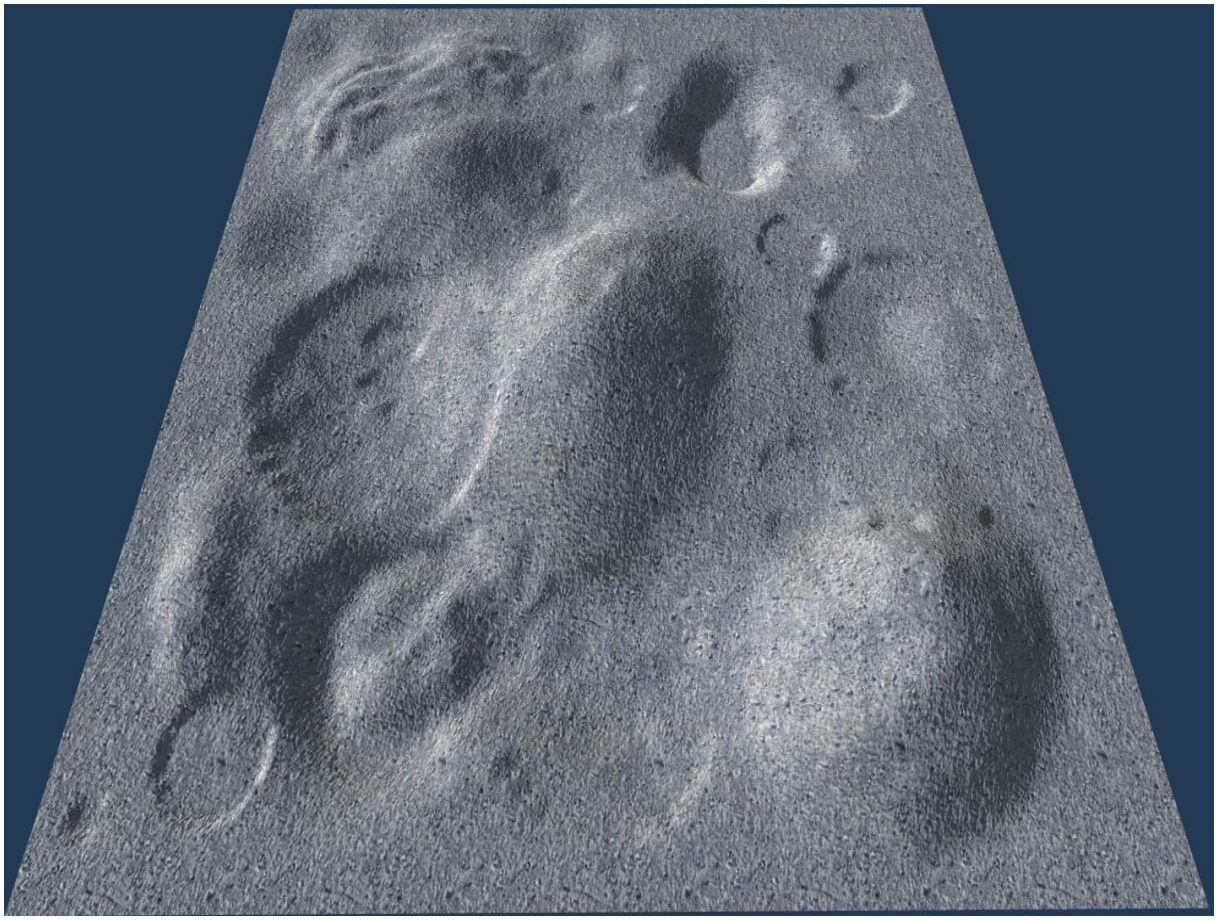


Рисунок 3.4 – Локація поверхні Луни (зроблено власноруч)

Я також вирішив, що ця локація підходить до гри, бо нашою тематикою є Sci-Fi, а зазвичай Sci-Fi йде пліч-о-пліч із космосом.

Другою локацією я вирішив зробити безжиттєву пустелю, яка приваблює своїми золотими пісочними пагорбами та таємничими кар’єрами, в яких можна або сховатися від загрози, або, випадково, зробити своєю гробницею. Приклад пустельної локації із кар’єром зображено на рисунку 3.5.

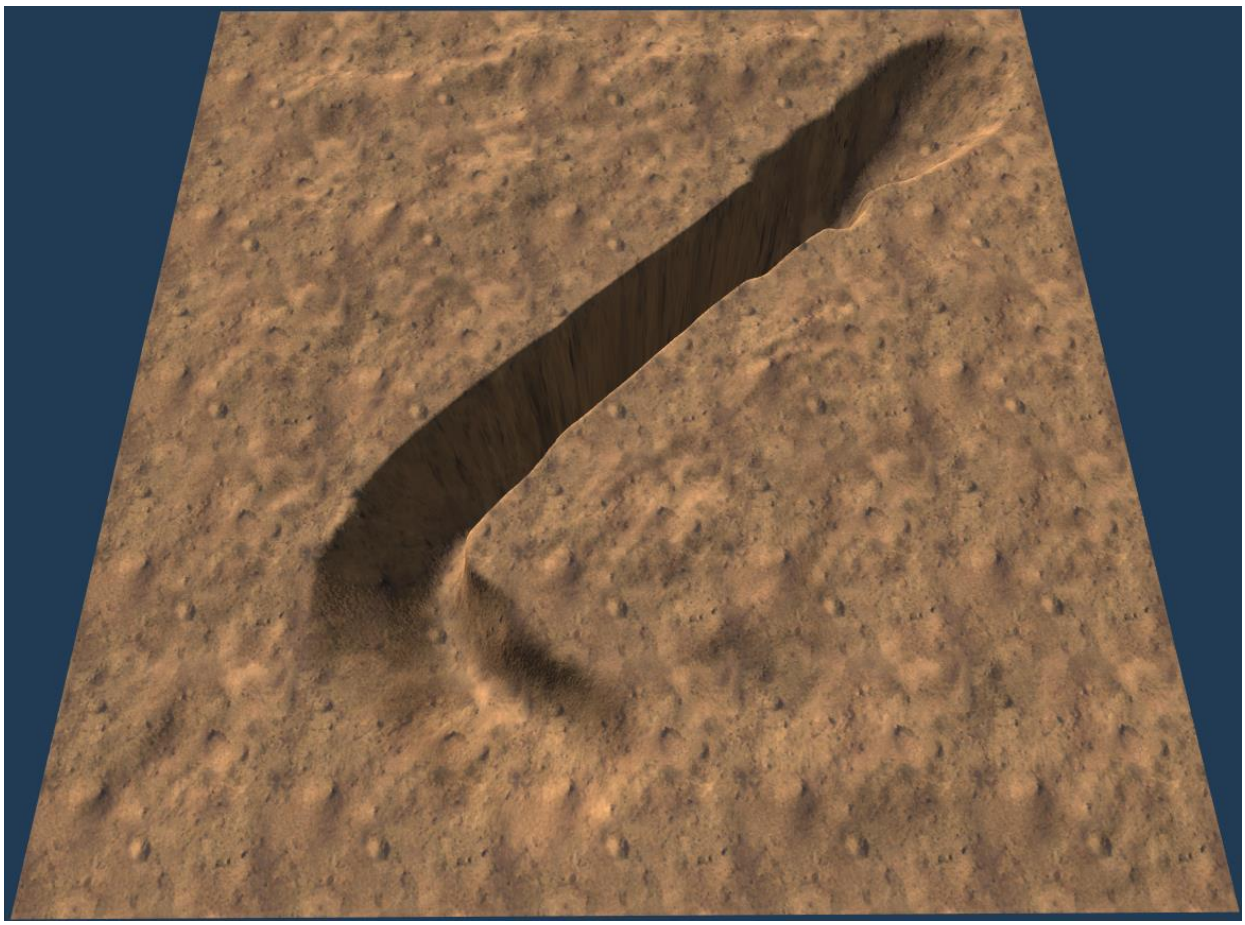


Рисунок 3.5 – Пустельна локація із таємничим кар’єром (зроблено власноруч)

Пустельна локація підходить до гри, бо в ній є багато вільного та відкритого простору, де можна спокійно пересуватися та відстрілюватися від супротивників, а також багато простору для фантазії, яким зробити той чи інший ландшафт. Важливо зберігати у гравців можливість входити і виходити з різних сторін кар’єрів та інших ігрових місць, що б гравець не застрягав та не порушував враження від гри.

Наступною локацією було вирішено зробити щось схоже на Земні пагорби, луга, поляни та інші зелені місця планети. Приклад ландшафту із зеленими пагорбами зображено на рисунку 3.6.

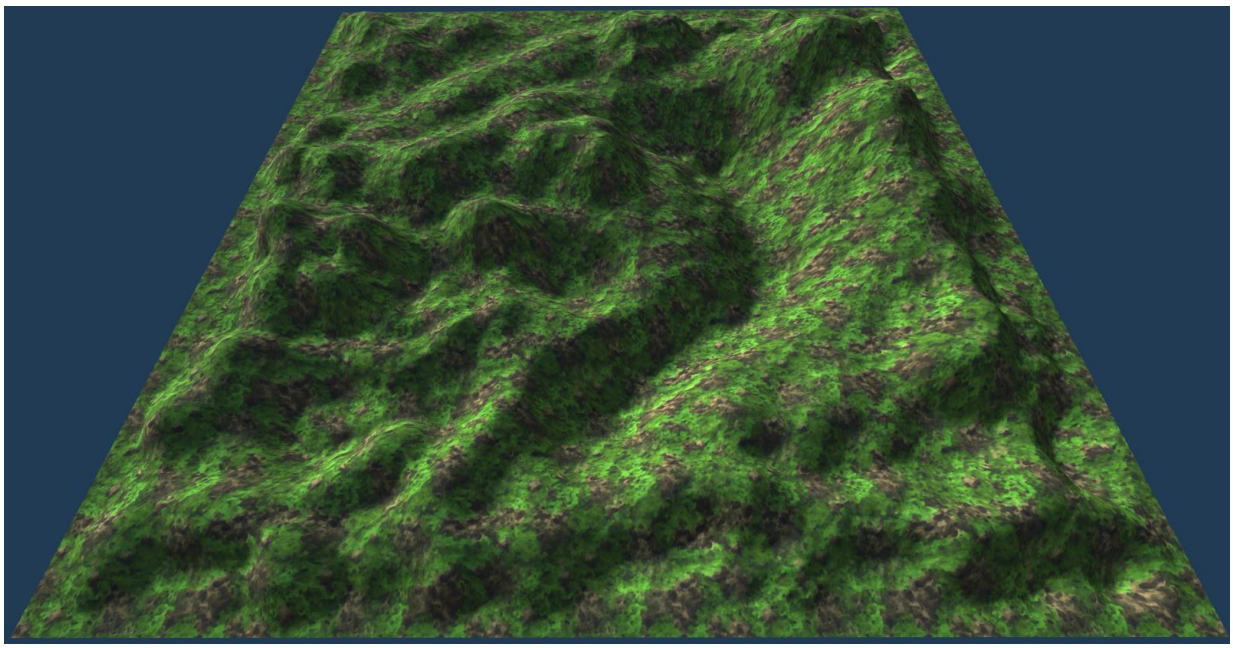


Рисунок 3.6 – Приклад локації із зеленими пагорбами (зроблено власноруч)

Рішення зробити зелену локацію обумовлено тим, що гравця потрібно утримувати зацікавленістю в дослідженні ігрового світу та різних ігрових місць, а також тим, що локації повинні максимально відрізнятися одне від одного. Зелена локація нагадує рідні місця для гравця, рідну планету та вимір, а також створює почуття чогось знайомого.

Отже, для проектування рівнів гри було обрано три різноманітні локації, які забезпечують захопливий ігровий досвід та відповідають тематиці Sci-Fi. Першою локацією стала поверхня Луни з її мальовничими пагорбами та кратерами, що додає космічної атмосфери. Другою локацією обрано безжиттєву пустелю з таємничими кар’єрами, що створюють простір для тактичних маневрів та прихованих небезпек. Третьою локацією стали зелені пагорби, луги та поляни, що нагадують земні пейзажі та підтримують інтерес гравця до дослідження різноманітних місць у грі. Кожна локація має свої унікальні особливості, що збагачують ігровий процес та дозволяють гравцям зануритись у різні світи.

4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

4.1 Створення перших ландшафтів

Напочатку всього потрібно створити декілька ландшафтів вручну. Для цього потрібно на Сцені в Unity створити новий ландшафт (рис. 4.1) за допомогою контекстного меню в ієрархії.

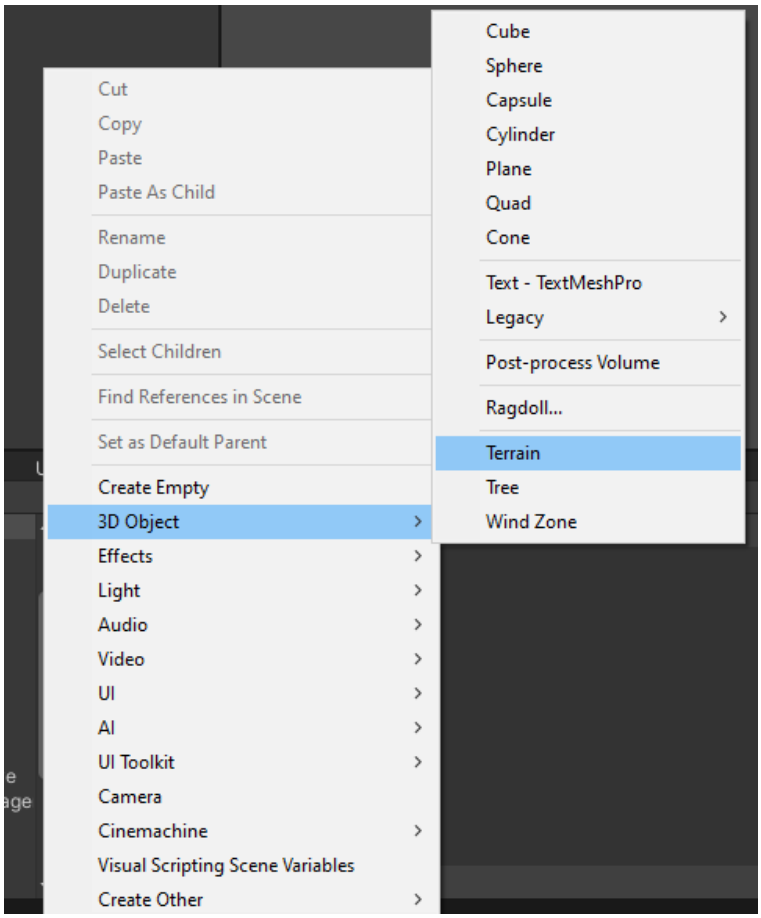


Рисунок 4.1 – Створення нового ландшафту (зроблено власноруч)

Після того, як на Сцені з’явився новий ландшафт, потрібно вибрати його та відкрити його налаштування. У вікні, що відкриється, потрібно виставити значення його ширини та довжини на 100 або на стільки, скільки потрібно (рис. 4.2).

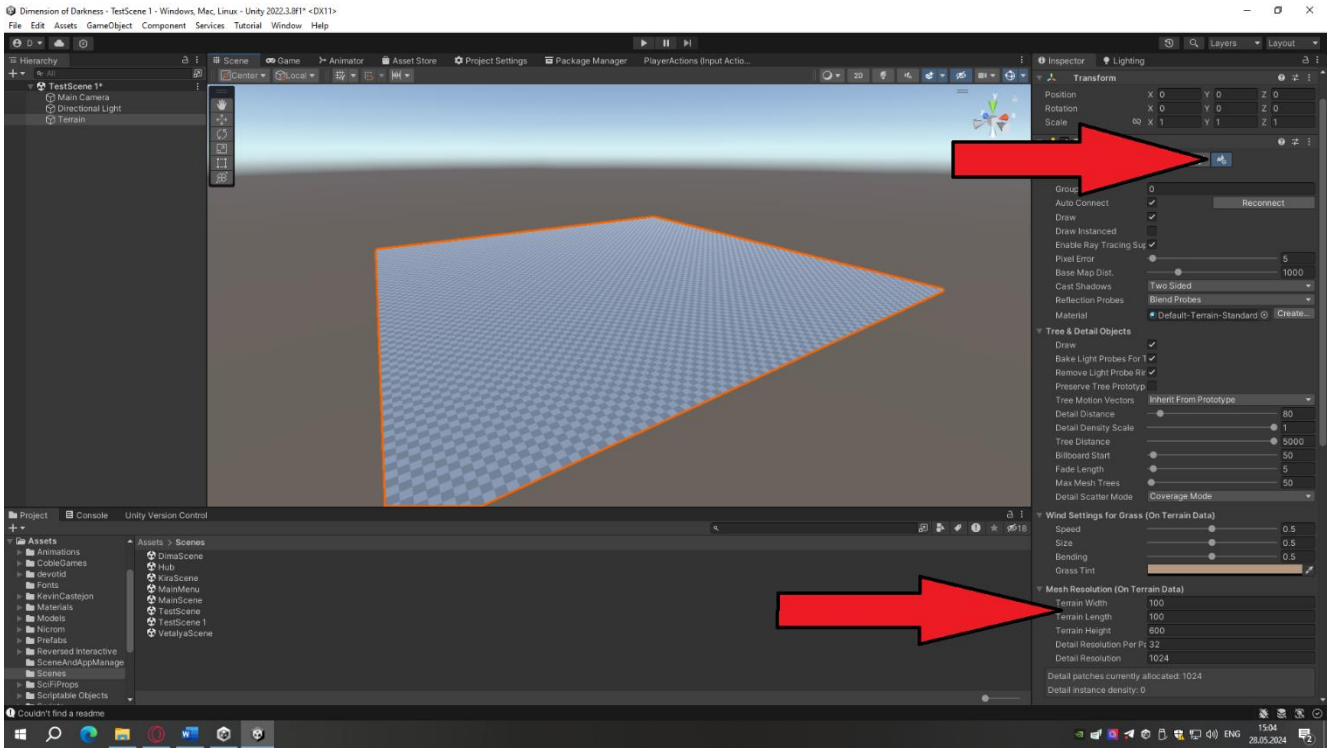


Рисунок 4.2 – Виставлення необхідної широти та довжини (зроблено власноруч)

Далі потрібно, за допомогою вбудованого пензля, створити красивий ландшафт, який підійшов би до локації, над якою ми хочемо зараз працювати. Наприклад, створимо поверхню Луни. Натискаємо на paint terrain, після чого у дропдаун меню обираємо set height для того, що б встановити середню висоту нашого ландшафту (рис. 4.3).

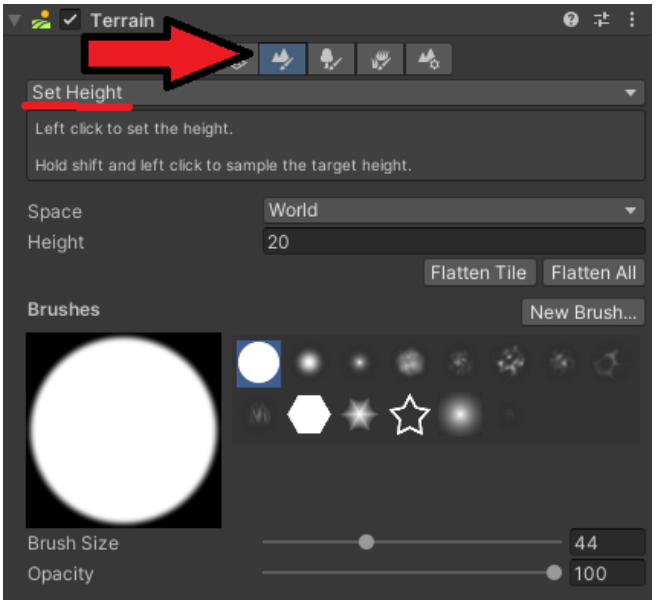


Рисунок 4.3 – Виставлення середньої висоти ландшафту (зроблено власноруч)

Обравши необхідне значення середньої висоти (для мене це 20), обираємо opacity 100, що б висота виставлялася одразу потрібна та brush size виставляємо по-більше. Наприклад, я взяв 44, як на рисунку 4.3. Тепер водимо пензлем по всьому нашому ландшафту, що б встановити висоту 20. Це потрібно для того, що б ми могли додавати усілякі кратери та вибоїни в землі.

Тепер нам необхідно додати текстуру Луни, для цього обираємо paint texture у дропдаун меню та натискаємо на edit terrain layers => add layer (рис. 4.4).

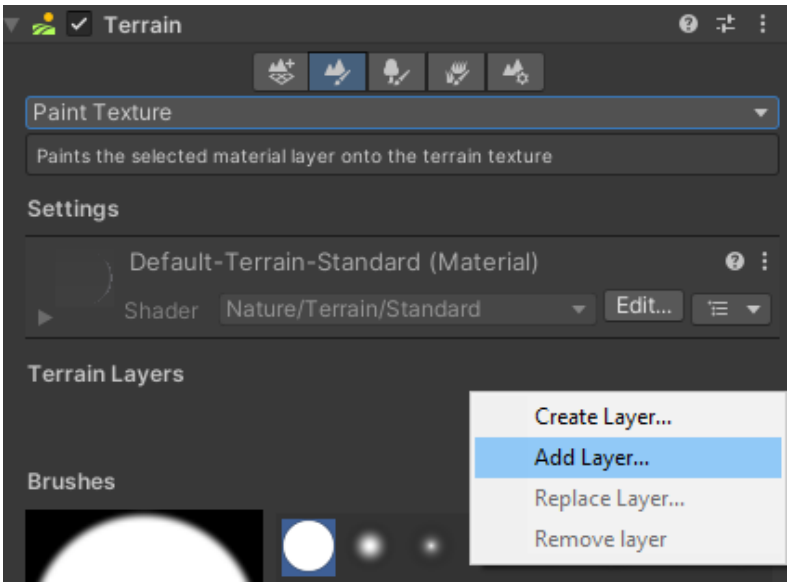


Рисунок 4.4 – Додавання слою текстури (зроблено власноруч)

Обираємо текстуру Луни та натискаємо на неї. Як ми бачимо, тепер на ландшафт накладено текстуру поверхні Луни (рис. 4.5).

Рисунок 4.5 – Ландшафт з текстурою поверхні Луни (зроблено власноруч)

Зараз нам потрібно обрати raise or lower terrain у дропдаун меню, виставити розмір та непрозорість як нам потрібно, обрати потрібний пензель та створити місцевість, яка буде максимально схожа на поверхню Луни. На рисунку 4.6 зображено готовий варіант ландшафту, який в мене вийшов.

Рисунок 4.6 – Ландшафт поверхні Луни (зроблено власноруч)

Після того, як ми завершили створення ландшафту поверхні Луни, необхідно зберегти його, як префаб, для того, що б потім мати змогу визивати його та звертатися до його компонентів. Створюємо папку Prefabs та зберігаємо його там. Для того, що б взаємодіяти з майбутньою випадковою генерацією ландшафтів, потрібно додати на цей префаб компонент Box Collider та скрипт генерації декорацій (рис 4.7), до якого ми ще повернемося.

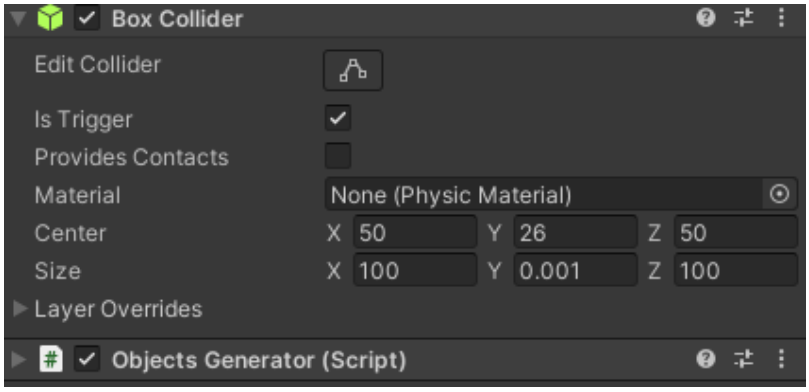


Рисунок 4.7 – Компоненти Box Collider та скрипт генерації декорацій (зроблено власноруч)

Наразі в нас є готовий префаб об’єкту ландшафту, який ми будемо використовувати як ігровий об’єкт.

4.2 Створення нових ландшафтів по шаблону існуючого

Коли ми створили декілька тематичних ландшафтів, нам потрібно збільшити їх кількість за допомогою обертання та віддзеркалення. Їх кількість збільшиться за формулою 4.1.

$$N = n * m * k \tag{4.1}$$

де N – кінцева кількість ландшафтів локації;

n – початкова кількість ландшафтів локації;

m – коефіцієнт кількісті разів обертання ландшафту, який дорівнює 4;

k – коефіцієнт кількісті разів віддзеркалення ландшафту, який дорівнює 2;

Код функції, що створює нову TerrainData та нові обернені копії ландшафтів:

```
private TerrainData GenerateTerrainData(Terrain exampleTerrain, int
variant)
{
    TerrainData terrainData = new TerrainData();
    TerrainData exampleTerrainData = exampleTerrain.terrainData;
    float[,] exampleTerrainHeights = exampleTerrainData.GetHeights(
        0,
        0,
        exampleTerrain.terrainData.heightmapResolution,
        exampleTerrain.terrainData.heightmapResolution);
    terrainData.heightmapResolution =
exampleTerrainData.heightmapResolution;


terrainData.SetDetailResolution(exampleTerrainData.detailResolution,
exampleTerrainData.detailResolutionPerPatch);

    if (variant == 1) terrainData.SetHeights(0, 0,
RotateMatrix(exampleTerrainHeights));
    else if (variant == 2) terrainData.SetHeights(0, 0,
MirrorMatrix(exampleTerrainHeights));
    terrainData.size = exampleTerrainData.size;
    return terrainData;
}

private T[,] RotateMatrix<T>(T[,] original)
{
```

```
    int width = original.GetLength(0);
    int height = original.GetLength(1);
    T[,] rotated = new T[width, height];

    for (int x = 0; x < width; x++)
    {
        for (int y = 0; y < height; y++)
        {
            rotated[y, width - 1 - x] = original[x, y];
        }
    }
    return rotated;
}
```

Після того, як ми обертаємо ландшафти, ми також повинні їх віддзеркалити. Ось код, який виконує віддзеркалення після обертання:

```
private T[,] MirrorMatrix<T>(T[,] original)
{
    int width = original.GetLength(0);
    int height = original.GetLength(1);
    T[,] mirrored = new T[width, height];
    for (int x = 0; x < width; x++)
    {
        for (int y = 0; y < height; y++)
        {
            mirrored[x, y] = original[y, x];
        }
    }
    return mirrored;
}
```

Щоб використати ці функції, ми повинні звернутися до них, попередньо створивши лист із усіма ландшафтами, які ми будемо використовувати. Код, який створює лист для усіх ландшафтів:

```
private void PickTerrainsAndCreateVersions()
{
    terrainGenerator = GetComponent<TerrainGenerator>();
    int randomIndex;
    if (terrainsIPick == 0 || terrainsIPick > terrainsArrays.Count)
        randomIndex = UnityEngine.Random.Range(0,
terrainsArrays.Count);
    else randomIndex = (int)terrainsIPick - 1;
```

```
    for (int i = 0, j = 0; i < terrainsArrays[randomIndex].Length;
i++, j++)
    {
        pickedTerrains.Add(terrainsArrays[randomIndex][i]);

pickedTerrains.Add(terrainGenerator.GenerateMirroredTerrain(pickedTer
rains[j]));
        j++;
        pickedTerrains[j].SetActive(false);
        pickedTerrains[j].transform.parent =
referenceObjects.transform;
        for (int k = 0; k < 3; k++)
        {

pickedTerrains.Add(terrainGenerator.GenerateRotatedTerrain(pickedTerr
ains[j]));
            j++;
            pickedTerrains[j].SetActive(false);
            pickedTerrains[j].transform.parent =
referenceObjects.transform;

pickedTerrains.Add(terrainGenerator.GenerateMirroredTerrain(pickedTer
rains[j]));
            j++;
            pickedTerrains[j].SetActive(false);
            pickedTerrains[j].transform.parent =
referenceObjects.transform;
        }
    }
}
```

Таким чином створюється список (лист) із усіма лпндшафтами потрібної локації

4.3 Виклик генерації ландшафтів

Спочатку потрібно створити необхідні масиви для наших локацій. Я назвав локацію з поверхнею Луни grayTerrain, з поверхнею пустелі yellowTerrain, а схожу з нашою Земною поверхнею greenTerrain. Створимо ці масиви:

```
public GameObject[] grayTerrains;
public GameObject[] greenTerrains;
public GameObject[] yellowTerrains;
```

Для того, що б випадково обрати одну з локацій, я створив лист з усіма

можливими локаціями та використав рефлексію для того, що б автоматично додати створені мною масиви ландшафтів та автоматизувати введення нових локацій у майбутньому:

```
private void CreateTerrainsArrays()
{
    FieldInfo[] fields = typeof(TilemapGenerator)
        .GetFields(BindingFlags.Instance | BindingFlags.Public);

    foreach (FieldInfo field in fields)
    {
        if (field.FieldType == typeof(GameObject[]))
        {
            terrainsArrays.Add((GameObject[])field.GetValue(this));
        }
    }
}
```

Коли ми обрали випадкову локацію, яка буде спавнитися у поточному забігу, нам необхідно розташувати перші дев’ять ландшафтів на карті. Для цього потрібно спочатку заповнити масив значеннями -1, окрім центрального місця цього масива:

```
for (int x = 0; x < width; x++)
{
    for (int y = 0; y < height; y++)
    {
        if (x == width / 2 && y == height / 2) gameObjectsToSpawn[x,
y] = 0;
        else gameObjectsToSpawn[x, y] = -1;
    }
}
```

Після того, як ми заповнили масив значень значеннями -1 та 0, необхідно заповнити масив випадковими значеннями, які відповідатимуть номеру ландшафту у листі обраних ландшафтів. Код, який заповнює масив випадковими значеннями:

```
bool isFinished = false;
for (int x = 1; x < width && !isFinished; x++)
{
    for (int y = 1; y < height && !isFinished; y++)
    {
```

```
            if (gameObjectsToSpawn[x, y] == -1) continue;

            int maxX = width - 1;
            int maxY = height - 1;

            if (x > 0 && gameObjectsToSpawn[x - 1,y] == -1)
                gameObjectsToSpawn[x - 1,y] = Random.Range(0,
    pickedTerrains.Count);
            if (y > 0 && gameObjectsToSpawn[x,y - 1] == -1)
                gameObjectsToSpawn[x,y - 1] = Random.Range(0,
    pickedTerrains.Count);
            if (x > 0 && y > 0 && gameObjectsToSpawn[x - 1,y - 1] == -1)
                gameObjectsToSpawn[x - 1,y - 1] = Random.Range(0,
    pickedTerrains.Count);
            if (x < maxX && gameObjectsToSpawn[x + 1,y] == -1)
                gameObjectsToSpawn[x + 1,y] = Random.Range(0,
    pickedTerrains.Count);
            if (y < maxY && gameObjectsToSpawn[x,y + 1] == -1)
                gameObjectsToSpawn[x,y + 1] = Random.Range(0,
    pickedTerrains.Count);
            if (x < maxX && y < maxY && gameObjectsToSpawn[x + 1,y + 1]
    == -1)
                gameObjectsToSpawn[x + 1,y + 1] = Random.Range(0,
    pickedTerrains.Count);
            if (x > 0 && y < maxY && gameObjectsToSpawn[x - 1,y + 1] == -
    1)
                gameObjectsToSpawn[x - 1,y + 1] = .Random.Range(0,
    pickedTerrains.Count);
            if (y > 0 && x < maxX && gameObjectsToSpawn[x + 1,y - 1] == -
    1)
                gameObjectsToSpawn[x + 1,y - 1] = Random.Range(0,
    pickedTerrains.Count);
            isFinished = true;
        }
    }
```

Останнім кроком потрібно заспавнити ці перші дев’ять ландшафтів. Для цього необхідно використати наступний код:

```
for (int x = 0; x < width; x++)
{
    for (int y = 0; y < height; y++)
    {
        if (gameObjectsToSpawn[x, y] == -1) continue;
        GameObject instance = Instantiate(
            pickedTerrains[gameObjectsToSpawn[x, y]],
            new Vector3(x-width/2, 0, y-height/2) * objectSize,
            Quaternion.identity,
```

```
                parentToSet.transform);
            instance.SetActive(true);
            instance.GetComponent<Terrain>().enabled = true;
            instance.GetComponent<ObjectsGenerator>().enabled = true;
        }
    }
```

Для того, що б викликати необхідний скрипт генерації наступних ландшафтів, необхідно, що б коллайдер на гравцеві перетнув BoxCollider на крайньому ландшафті. Після цього BoxCollider на поточному ландшафті видаляється та спавняться нові ландшафти навколо поточного, у тих місцях, де ще нема ландшафтів. Для цього на гравцеві висить скрипт TilemapGeneratorInPlayer:

```
public static Action<Vector3> OnEnteredIntoATile;

private void OnTriggerEnter(Collider tile)
{
    if (tile.gameObject.CompareTag("Terrain"))
    {
        OnEnteredIntoATile?.Invoke(tile.transform.position);
        Destroy(tile.gameObject.GetComponent<BoxCollider>());
    }
}
```

Цей скрипт викликає генерацію ландшафтів навколо поточного ландшафту, скрипт якої наведено у додатку Г.

4.4 Спавн об’єктів на створених ландшафтах

Після того, як на карті було заспавнено новий ландшафт, на ньому будуть випадково спавнитися декорації. Код, який випадково спавнить декорації, в залежності від налаштувань, по всій площі поточного ландшафта:

```
private float objectSize;

public List<GameObject> gameObjectPrefabs;

public int minObjectsPerTile = 1;
public int maxObjectsPerTile = 1;

public void Start()
```

```
{
    if(gameObjectPrefabs.Count == 0) return;

    Terrain terrain = transform.GetComponent<Terrain>();
    objectSize = terrain.terrainData.size.x;
    for(int i = Random.Range(minObjectsPerTile, maxObjectsPerTile+1),
k = 0; k<i; k++)
    {
        float randPositionX = Random.Range(0, objectSize);
        float randPositionZ = Random.Range(0, objectSize);
        float positionY = terrain.SampleHeight(new Vector3(
            transform.position.x +randPositionX,
            0,
            transform.position.z + randPositionZ));
        Vector3 position = new Vector3(
            randPositionX,
            positionY,
            randPositionZ);
        Instantiate(
            gameObjectPrefabs[Random.Range(0,
gameObjectPrefabs.Count)],
            position + terrain.transform.position,
            Quaternion.Euler(0, Random.Range(0f, 360f), 0),
            transform);
    }
    Destroy(this);
}
```

Кінцевий варіант перших дев’яти ландшафтів зображено на рисунку 4.8.

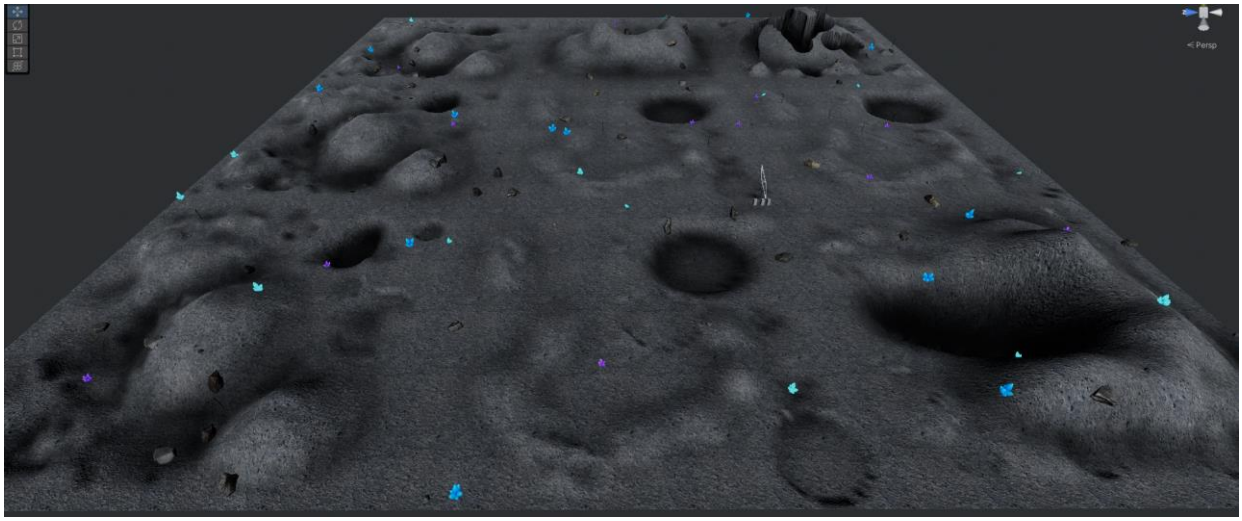


Рисунок 4.8 – Випадково заспавнені 9 ландшафтів із декораціями

Як можна побачити, тепер наша локація із декораціями виглядає дуже красиво та цікаво для дослідження гравцем.

5 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 Тестування ігрового застосунку

Ігровий програмний застосунок був протестований відповідно до заздалегідь розробленого тест-плану. Нижче наведено Mind Map, яка використовується для нашого тест-плану (рис. 5.1). Тестування було проведено за методологією Blackbox, або ж, інакше, за допомогою «чорної скриньки».

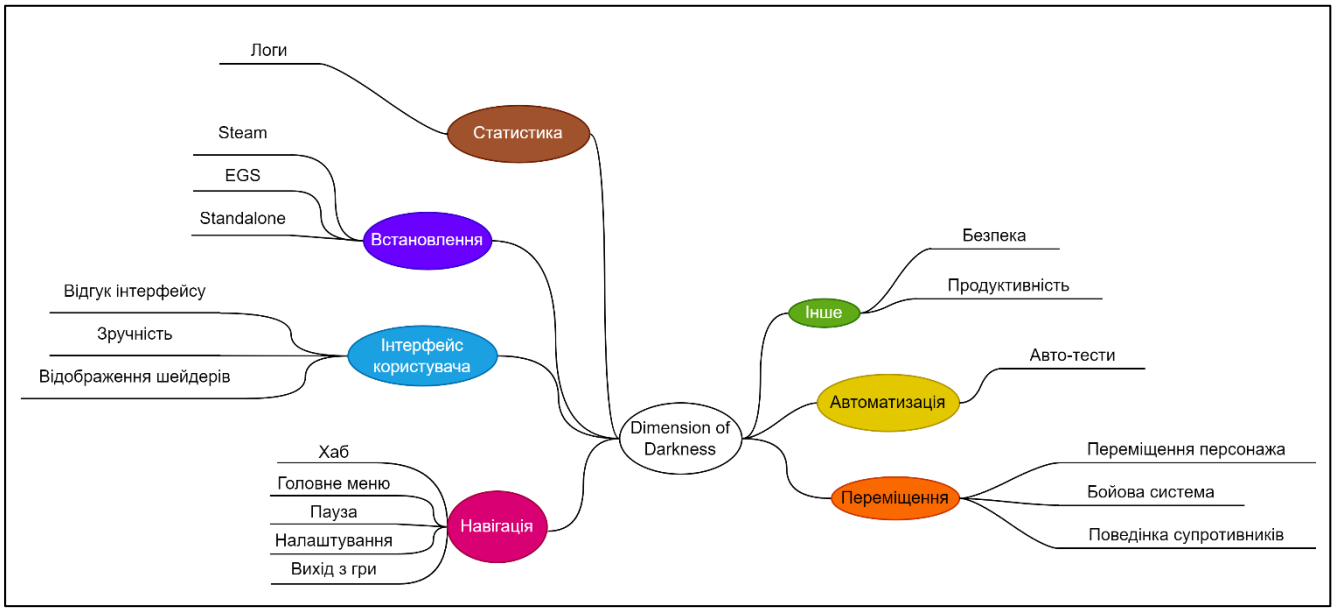


Рисунок 5.1 – Mind Map для гри «Dimension of Darkness» (зроблено власноруч)

За допомогою цієї карти ми можемо поділити тестування проекту на окремі частини:

а) Статистика;

б) Встановлення;

в) Навігація;

г) Інтерфейс користувача;

д) Автоматизація;

е) Переміщення;

ж) Інше.

Було проведено тестування в процесі розробки додатку. Тестування функціональності, навігації та інтерфейсу користувача.

5.2 Розробка тестових випадків

Виходячи з класифікації, представленої у вигляді діаграми Mind Map, був розроблений набір тестових сценаріїв для всебічного тестування системи. Кожен тестовий сценарій включає в себе опис обставин, передбачуваний і фактичний результати, а також додаткові зауваження. У таблицях 5.1–5.8 представлено тестові сценарії, які були складені в ході тестування програмного забезпечення. Пріоритетність тестового сценарію визначається як найвища з оцінкою P1 і найнижча з оцінкою P4. Ступінь критичності можливої помилки оцінюється на шкалі від S1 (мінімальна критичність) до S4 (критичність, що може зруйнувати геймплей).

Таблиця 5.1 – Баг 1

Назва багу	Неправильне поводження VALERA
Короткий опис	При слідкуванні за гравцем робот смикається
Компонент додатку	VALERA (VALERA Controller)
Важливість	S2 Висока
Пріоритет	P2 Середній
Кроки відтворення	1. Пройти гравцем у будь-який бік
Фактичний результат	Рухи робота перериваються
Очікуваний результат	Робот плавно слідкує за гравцем

Таблиця 5.2 – Баг 2

Назва багу	Об'єкти спавняться за рельєфами
Короткий опис	Деякі об’єкти, які повинні бути на рельєфах, які їх спавнять, спавняться за ними
Компонент додатку	Objects Generator
Важливість	S3 Середня
Пріоритет	P2 Середній
Кроки відтворення	1. Вийти у вікно «Сцена» 2. Натиснути на будь-який рельєф
Фактичний результат	Деякі об’єкти спавняться за їхніми рельєфами
Очікуваний результат	Усі об’єкти спавняться в межах того рельєфу, який їх створює

Як ми бачимо у таблиці 5.1, робот VALERA поводить себе неприродньо та некрасиво, порушується враження того, що гравець десь у іншому вимірі та звертає увагу на недоліки гри.

Баг 2, з таблиці 5.2, звісно, не заважає гравцю, але створює зайве навантаження на систему.

Таблиця 5.3 – Баг 3

Назва багу	Проходження скрізь скелі
Короткий опис	Гравець може проходити крізь тверді скелі
Компонент додатку	Collider
Важливість	S1 Блокуюча
Пріоритет	P1 Високий
Кроки відтворення	1. Знайти будь-яку скелю 2. Спробувати пройти крізь неї
Фактичний результат	Гравець може пройти крізь скелю
Очікуваний результат	Гравець не може пройти крізь скелю

Таблиця 5.4 – Баг 4

Назва багу	Надто сильне освітлення VALERA
Короткий опис	Якщо світити вниз на гравця, освітлення надто потужне
Компонент додатку	Flashlight (Light)
Важливість	S3 Середня
Пріоритет	P3 Низький
Кроки відтворення	1. Посвітити вниз на гравця
Фактичний результат	Гравець надто засвітлений
Очікуваний результат	Освітлення природне

У багу 3, з таблиці 5.3, знайдено типовий баг ігрових додатків, коли гравець може проходити крізь тверді об’єкти. Зазвичай, це порушує ігрове враження.

Таблиця 5.5 – Баг 5

Назва багу	Освітлення скель
Короткий опис	Скелі неприродно освітлюються
Компонент додатку	Flashlight (Light)
Важливість	S2 Висока
Пріоритет	P2 Середній
Кроки відтворення	1. Знайти скелю 2. Посвітити на неї 3. Потихеньку відводити ліхтарик в сторону
Фактичний результат	Скелі різко освітлюються та затемнюються
Очікуваний результат	Освітлюється та частина скелі, на яку світять

Таблиця 5.6 – Баг 6

Назва багу	Тіні на самому ландшафті неприродні
Короткий опис	Карта нормалей на ландшафтах створює неприродні тіні
Компонент додатку	Texture (Normal map)
Важливість	S2 Висока
Пріоритет	P2 Середній
Кроки відтворення	1. Освітити підлогу ліхтариком
Фактичний результат	Тіні надто потужні
Очікуваний результат	Тіні природні

Як можна побачити, баги у таблицях 5.5 – 5.6 пов’язані зі світлом, яке дуже сильно сприяє враженню від гри у цілому. Робота із правильним поводженням світла дуже важлива при розробці нашого додатку, адже невірне освітлення може вплинути на загальну атмосферу та реалістичність сцен. Виправлення цих багів має стати пріоритетом для команди, щоб забезпечити найкращий ігровий досвід. Усунення цих проблем не лише покращить візуальну якість гри, але й допоможе уникнути розчарувань гравців, пов’язаних з технічними недоліками. Потрібно забезпечити гладке та реалістичне освітлення, що є ключовим для створення занурюючого ігрового середовища, яке відіграє дуже важливу роль у загальному сприйнятті гри.

Таблиця 5.7 – Баг 7

Назва багу	На ландшафти не падають тіні
Короткий опис	Тіні не відображаються на ландшафтах
Компонент додатку	Terrain
Важливість	S3 Середня
Пріоритет	P2 Середній
Фактичний результат	Тіней нема
Очікуваний результат	Тіні є

Таблиця 5.8 – Баг 8

Назва багу	Потужний ліхтарний стовп
Короткий опис	Ліхтарний стовп може створювати надпотужне освітлення
Компонент додатку	Flashing (Lighter-> Spot Light)
Важливість	S3 Середня
Пріоритет	P2 Середній
Кроки відтворення	1. Знайти ліхтарний стовп на сірій локації 2. Чекати, доки ліхтар не почне світити надто неприродньо-потужно
Фактичний результат	Неприродньо-потужне освітлення
Очікуваний результат	Освітлення природне, є максимум потужності

У таблиці 5.7 зазначається, що у грі виявлено проблему з відображенням тіней на ландшафтах. Помилки призводять до відсутності тіней, що роблять сцену менш реалістичною та знижують якість візуального досвіду. Очікувано, що тіні мають коректно проектуватися на ландшафт.

У таблиці 5.8 зазначено надмірно потужне освітлення від ліхтарних стовпів, що створює неприродно яскраве світло. Це може порушувати атмосферу гри та відволікати гравців. Очікується, що освітлення від ліхтарів буде природним та не перевищуватиме встановленого максимуму потужності, забезпечуючи комфортне сприйняття сцени.

6 ВПРОВАДЖЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

6.1 Соціальне впровадження проекту

У процесі розробки нашого застосунку ми також створили сторінки в соціальних мережах для просування проекту. Ми використовували такі платформи, як Telegram, YouTube, Instagram та Twitter/X. Найкращі результати були отримані на платформі Telegram, оскільки вона є найпопулярнішою серед нашої цільової аудиторії. Instagram та YouTube також були ефективними, дозволяючи нам залучати нових користувачів. З іншого боку, Twitter (або X) не були настільки ефективними, оскільки наша цільова аудиторія майже не користується цими платформами.

У майбутньому ми плануємо продовжити використовувати YouTube, Telegram та Instagram для подальшого просування нашої гри. Також розглядаємо можливість створення сторінок на спеціалізованих ресурсах, таких як Steam Community або Fandom.

6.2 Реліз гри на платформі Steam

Один із ключових етапів нашого проекту – це випуск гри на платформі Steam. Це рішення обумовлене популярністю Steam серед геймерів у всьому світі, а також зручними інструментами для розробників, які платформа надає. Steam дозволяє нам охопити велику аудиторію та забезпечити гравцям простий доступ до нашої гри.

Для випуску гри на Steam ми використовуємо модель Standalone, яка дозволяє гравцям завантажувати та запускати гру без потреби у додаткових програмних компонентах. Ми впевнені, що це підвищить зручність використання та приверне більше користувачів. Крім того, ми плануємо інтегрувати нашу гру з екосистемою Steam Community, де користувачі зможуть обговорювати гру, ділитися враженнями та отримувати підтримку.

6.3 Реліз гри на платформі Epic Games Store (EGS)

Окрім Steam, ми також плануємо випуск гри на платформі Epic Games Store (EGS). Це ще одна популярна платформа серед геймерів, яка дозволяє охопити додаткову аудиторію та збільшити продажі гри. EGS пропонує вигідні умови для розробників, зокрема, нижчу комісію порівняно з іншими платформами, що робить її привабливою для нас як для розробників.

EGS також підтримує модель Standalone, що дозволяє гравцям легко завантажувати та запускати гру. Ми плануємо скористатися всіма можливостями цієї платформи для залучення нових користувачів та створення активної спільноти гравців.

6.4 Подальші плани та розвиток

Ми маємо амбітні плани щодо розвитку нашої гри. Після релізу на Steam ми плануємо регулярно випускати оновлення з новими функціями, рівнями та виправленнями помилок. Також розглядаємо можливість випуску гри на інших платформах, таких як Epic Games Store та консолі.

Ми продовжимо активно просувати наш проект у соціальних мережах та на спеціалізованих платформах, підтримуючи інтерес до гри та залучаючи нових користувачів. Наша мета – створити активну та зацікавлену спільноту гравців, які будуть підтримувати та розвивати гру разом з нами.

Таким чином, ми впевнені, що наш проект має великі перспективи і зможе зайняти гідне місце на ринку комп'ютерних ігор.

ВИСНОВКИ

У ході виконання цієї кваліфікаційної роботи бакалавра було розроблено систему генерації ландшафту та декорацій для демонстраційного ігрового застосунку у жанрі shoot’em up. Розробка велася на базі движка Unity3D 2022, з використанням мови програмування C# та підтримкою ОС Windows 10/11.

Було проведено аналіз предметної галузі, що включав вивчення існуючих ігор з аналогічними жанрами та механіками, а також аналіз ринку відеоігор для визначення поточних трендів та вподобань гравців. Це дозволило сформувати чіткі вимоги до системи генерації, які відповідають сучасним стандартам ігрової індустрії.

На основі аналізу було розроблено архітектуру системи, яка включає модульність та легкість налаштування параметрів генерації. Було створено набір діаграм, які ілюструють взаємодію компонентів системи та їх інтеграцію з іншими елементами ігрового застосунку.

Система генерації ландшафту та декорацій була оптимізована для забезпечення високої продуктивності та якості візуальних ефектів. Вона здатна генерувати різноманітні ландшафти, що сприяють геймплею та зануренню гравців у ігровий світ.

Система розроблена з можливістю легкої модифікації та розширення, що дозволяє адаптувати її до майбутніх вимог та змін у жанрі. Це включає можливість додавання нових типів декорацій, зміну параметрів генерації та інтеграцію з новими ігровими механіками.

З урахуванням викладеного вище, можна зробити висновок, що розроблена система генерації ландшафту та декорацій для ігрового застосунку у жанрі shoot’em up є ефективною, гнучкою та здатною задовольнити потреби сучасних ігор та їх розробників. Система відкриває широкі можливості для творчості та інновацій у дизайні ігрових світів, роблячи її цінним інструментом для ігрової індустрії.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Shoot’em up – explanation [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Shoot_%27em_up (дата звернення: 13.05.2024).

2. Space Invaders (1978) [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Space_Invaders (дата звернення: 13.05.2024).

3. Terraria [Електронний ресурс] – Режим доступу до ресурсу: https://store.steampowered.com/app/105600/Terraria/?l=ukrainian (дата звернення: 13.05.2024).

4. Minecraft – офіційна сторінка гри [Електронний ресурс] – Режим доступу до ресурсу: https://www.minecraft.net (дата звернення: 13.05.2024).

5. Starbound [Електронний ресурс] – Режим доступу до ресурсу: https://store.steampowered.com/app/211820/Starbound/ (дата звернення: 13.05.2024).

6. No Man’s Sky [Електронний ресурс] – Режим доступу до ресурсу: https://store.steampowered.com/app/275850/No_Mans_Sky/?l=ukrainian (дата звернення: 13.05.2024).

7. Як підвищити FPS в іграх [Електронний ресурс] – Режим доступу до ресурсу: https://www.moyo.ua/news/kak-povysit-fps-v-igrakh-6-sovetov-i-5-programm.html (дата звернення: 13.05.2024).

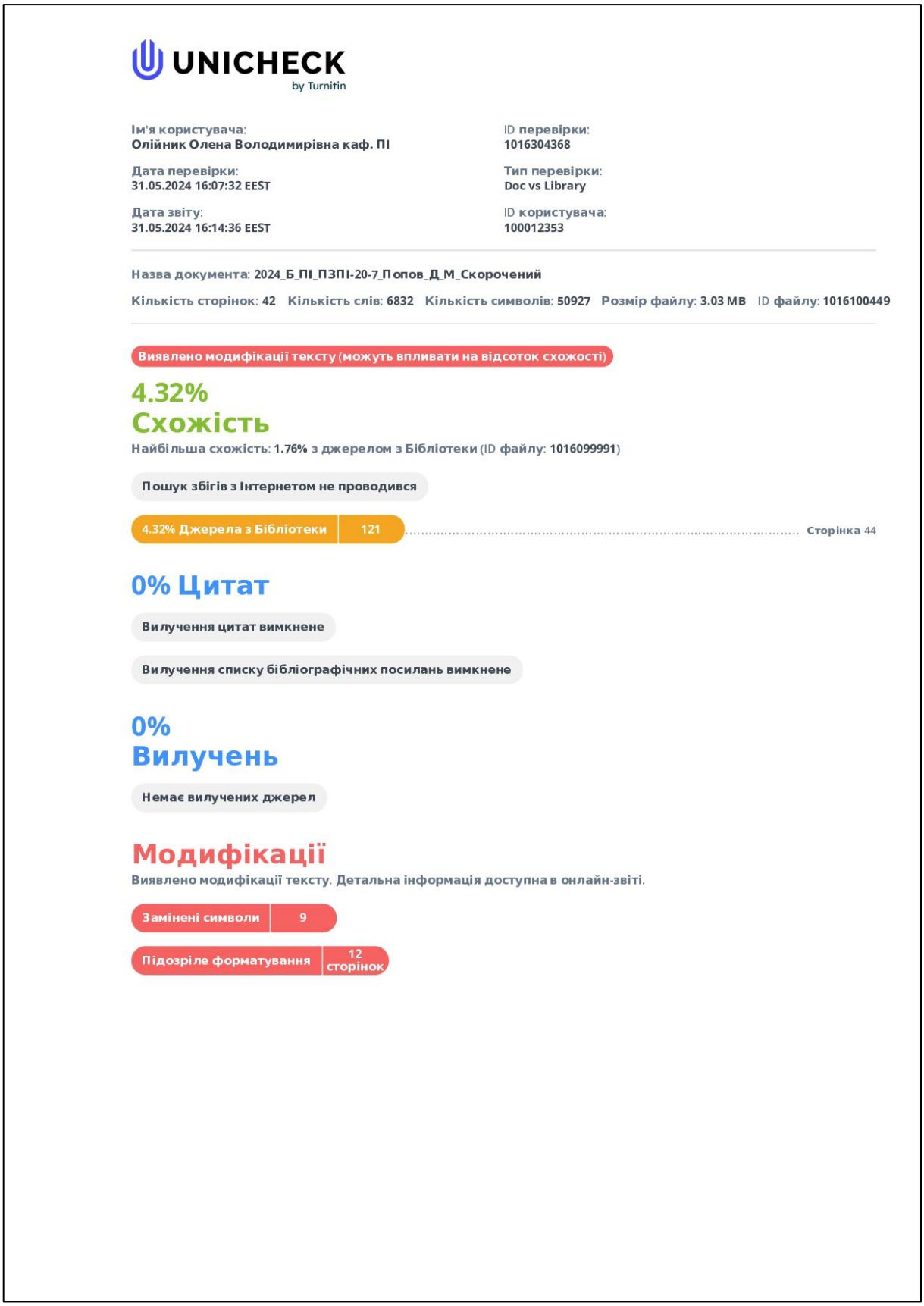
8. Що таке шум Перліна та як його використовувати при створенні ігор [Електронний ресурс] – Режим доступу до ресурсу: https://gamedev.dou.ua/articles/mathematics-gamedev-perlin-noise/ (дата звернення: 13.05.2024).

9. Процедурна генерація світу за допомогою Шуму Перліна [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/companies/selectel/articles/731506/ (дата звернення: 13.05.2024).

10. МЕТОДИ ТА ПРОГРАМНІ ЗАСОБИ ГЕНЕРАЦІЇ ЛАНДШАФТУ В ІГРОВИХ ПРОГРАМАХ [Електронний ресурс] – Режим доступу до ресурсу: https://ela.kpi.ua/server/api/core/bitstreams/52acd83b-cdac-4bf9-83cf-06933212b559/content (дата звернення: 13.05.2024).

ДОДАТОК А

Звіт результатів перевірки на унікальність тексту в базі ХНУРЕ



UNICHECK
by Turnitin

Ім'я користувача:
Олійник Олена Володимирівна каф. ПІ

ID перевірки:
1016304368

Дата перевірки:
31.05.2024 16:07:32 EEST

Тип перевірки:
Doc vs Library

Дата звіту:
31.05.2024 16:14:36 EEST

ID користувача:
100012353

Назва документа: 2024_Б_ПІ_ПЗПІ-20-7_Попов_Д_М_Скорочений

Кількість сторінок: 42 Кількість слів: 6832 Кількість символів: 50927 Розмір файлу: 3.03 MB ID файлу: 1016100449

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

4.32%
Схожість

Найбільша схожість: 1.76% з джерелом з Бібліотеки (ID файлу: 1016099991)

Пошук збігів з Інтернетом не проводився

4.32% Джерела з Бібліотеки 121 ..... Сторінка 44

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи 9

Підозріле форматування 12 сторінок

Рисунок А.1 – Звіт Unicheck

ДОДАТОК Б

Слайди презентації

Рисунок Б.1 – Слайд 1

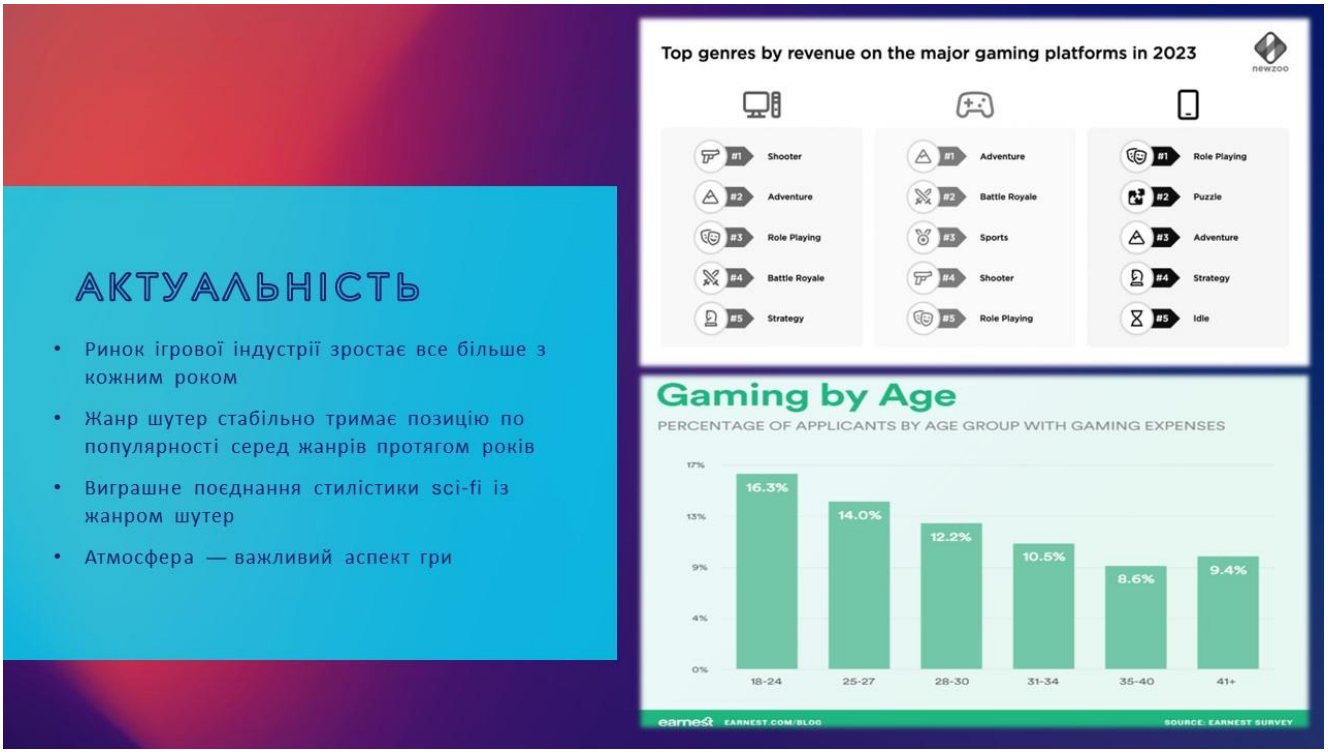


Рисунок Б.2 – Слайд 2

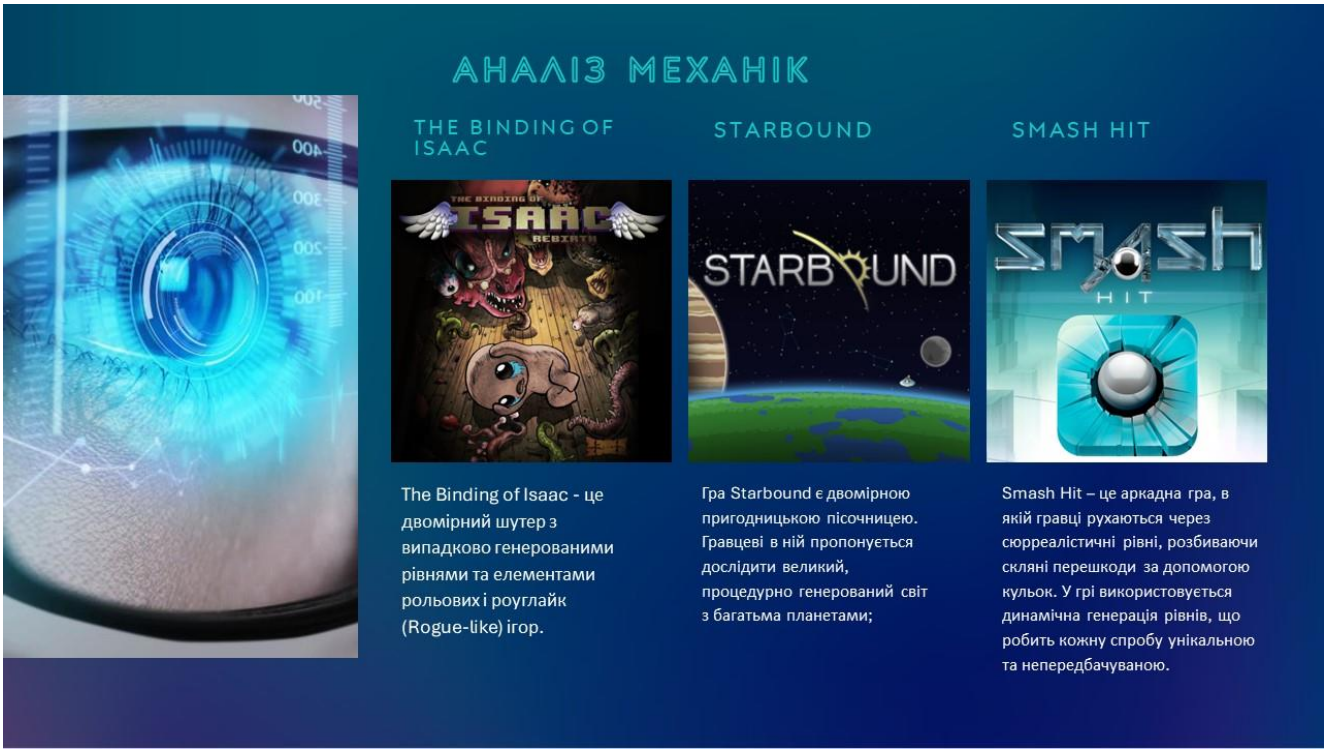


Рисунок Б.3 – Слайд 3

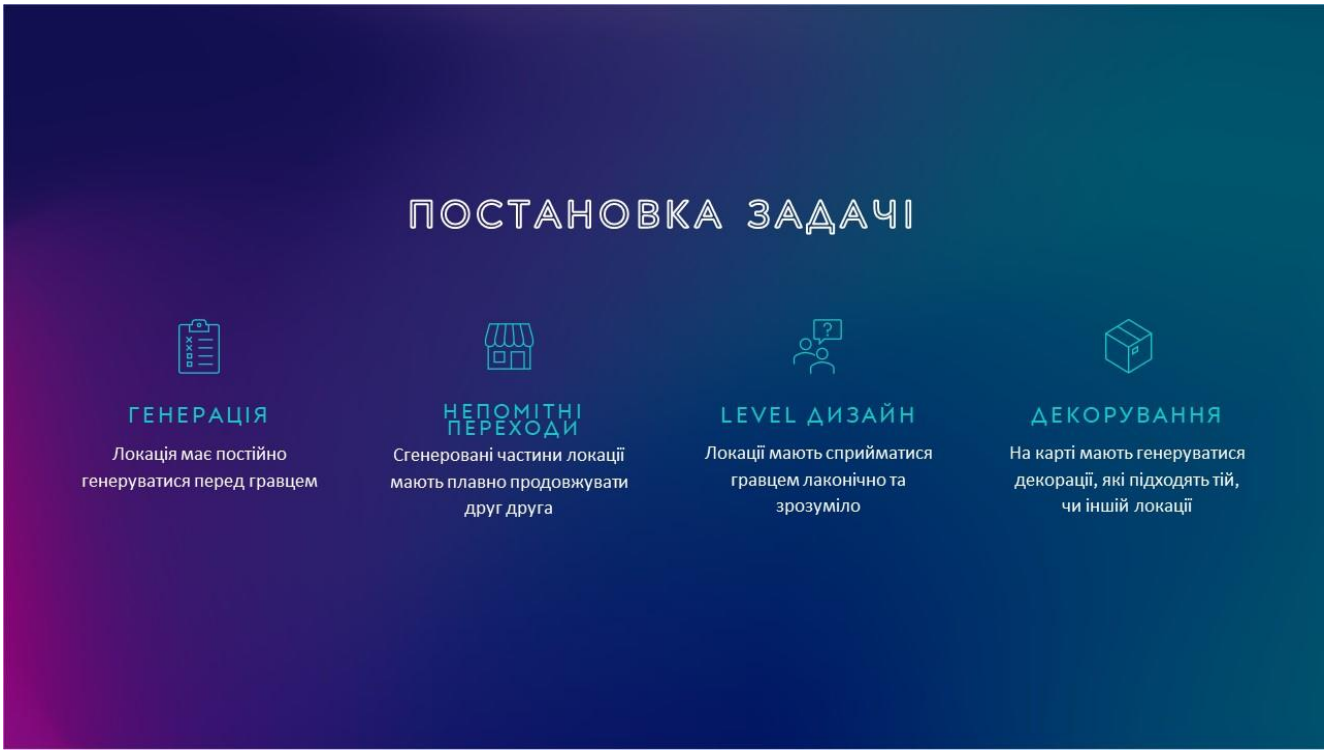


Рисунок Б.4 – Слайд 4

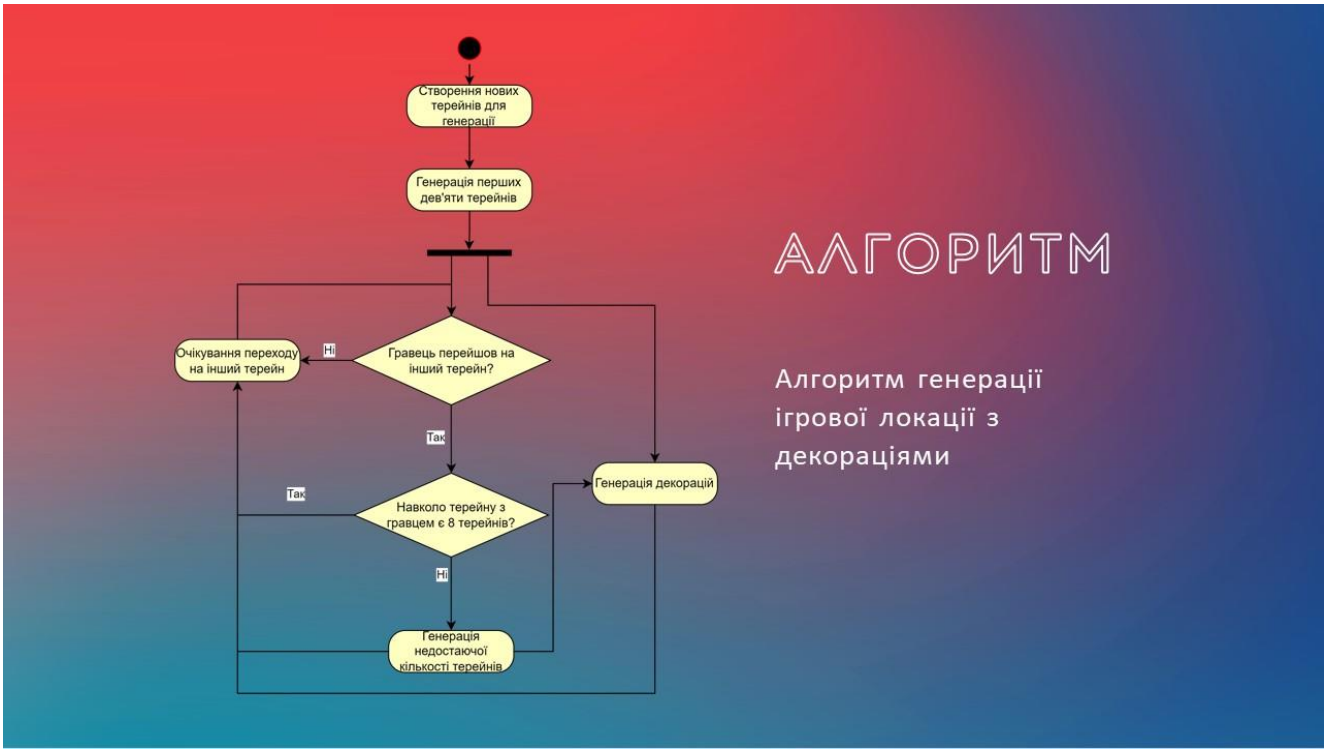


Рисунок Б.5 – Слайд 5

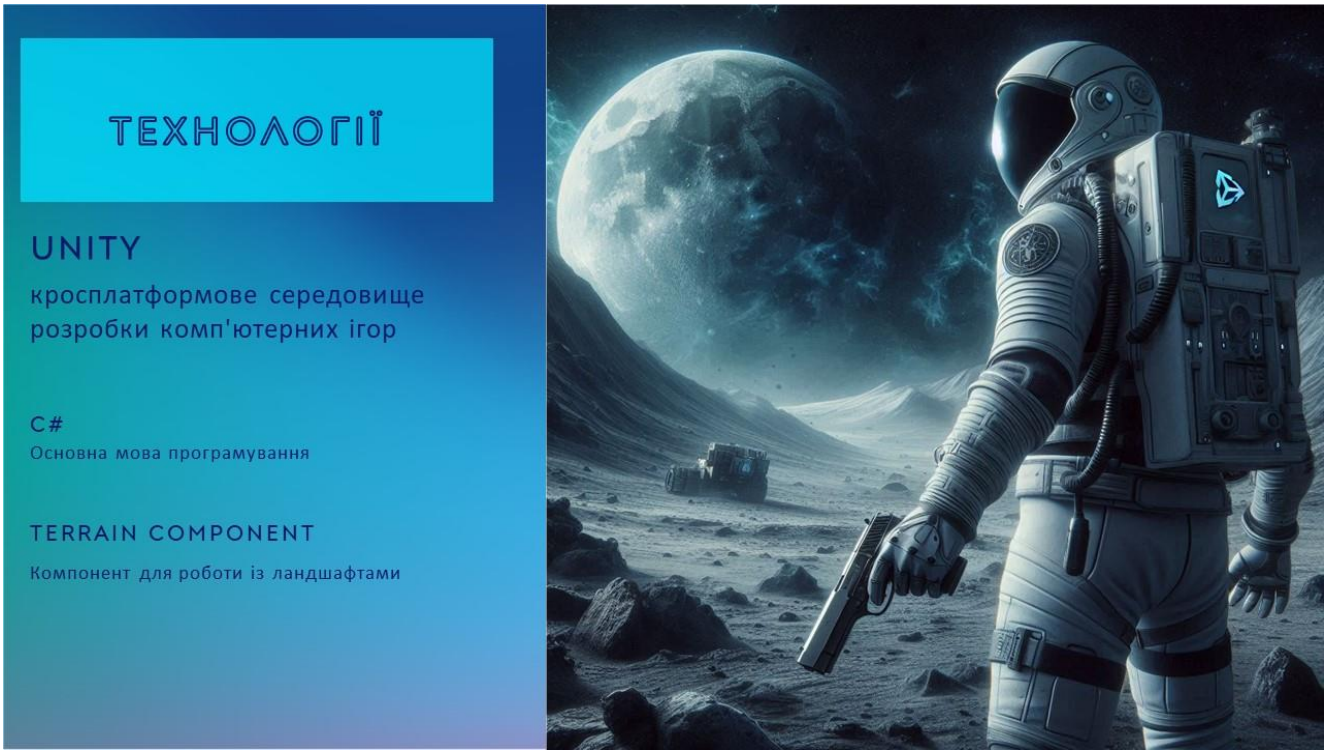


Рисунок Б.6 – Слайд 6

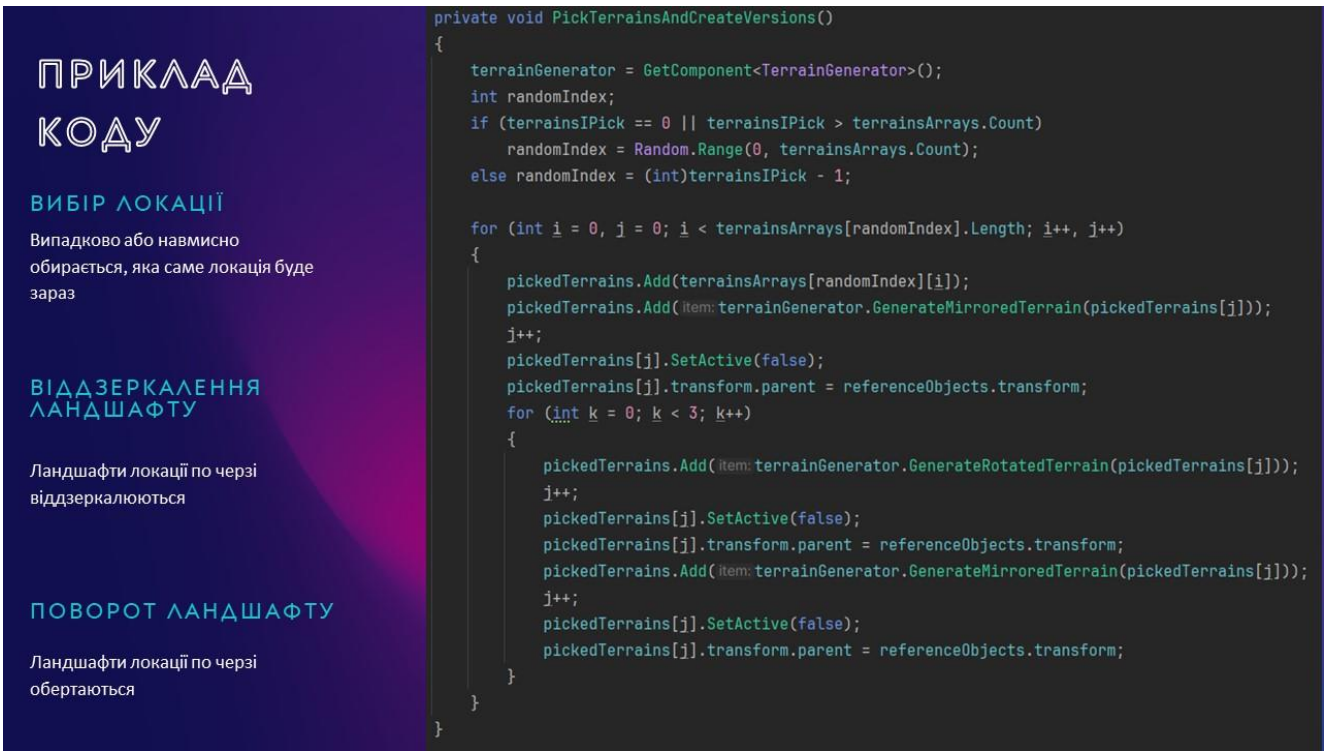


Рисунок Б.7 – Слайд 7

Рисунок Б.8 – Слайд 8

Рисунок Б.9 – Слайд 9

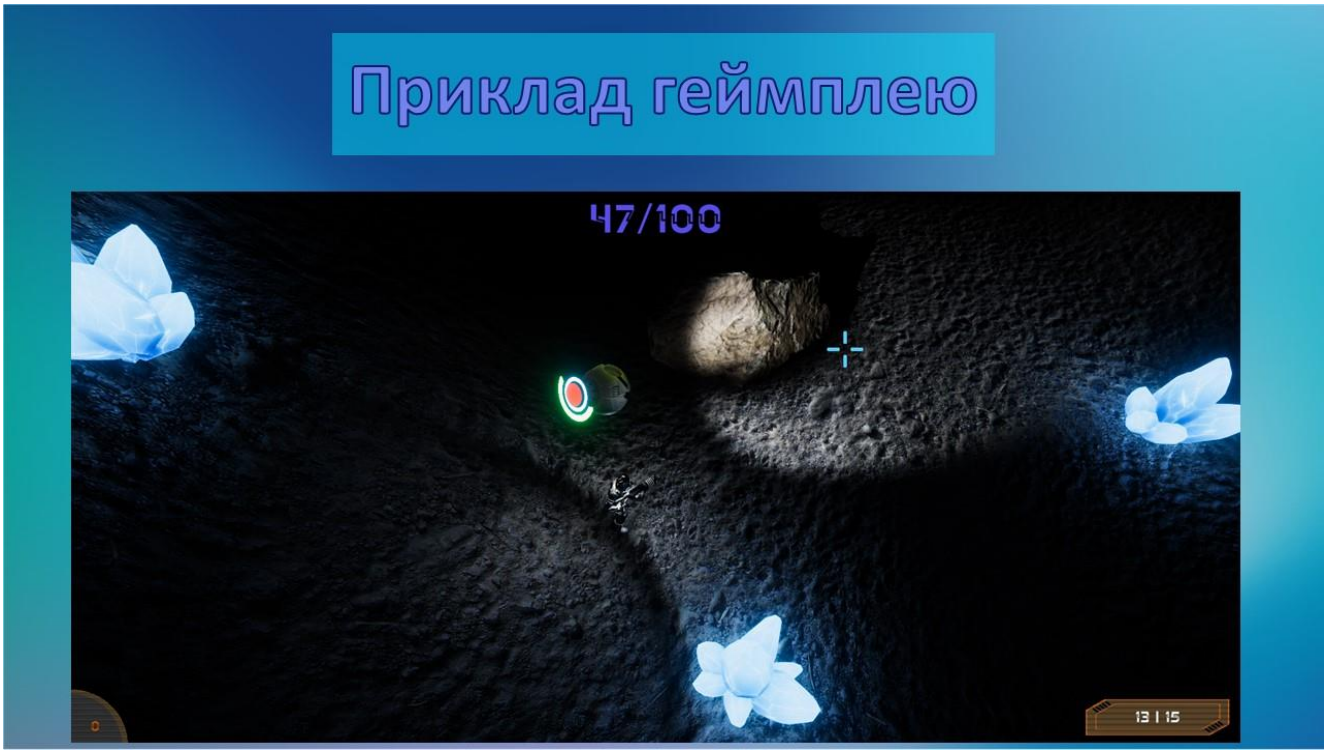


Рисунок Б.10 – Слайд 10

Рисунок Б.11 – Слайд 11

ДОДАТОК В

Концепт-документ до гри

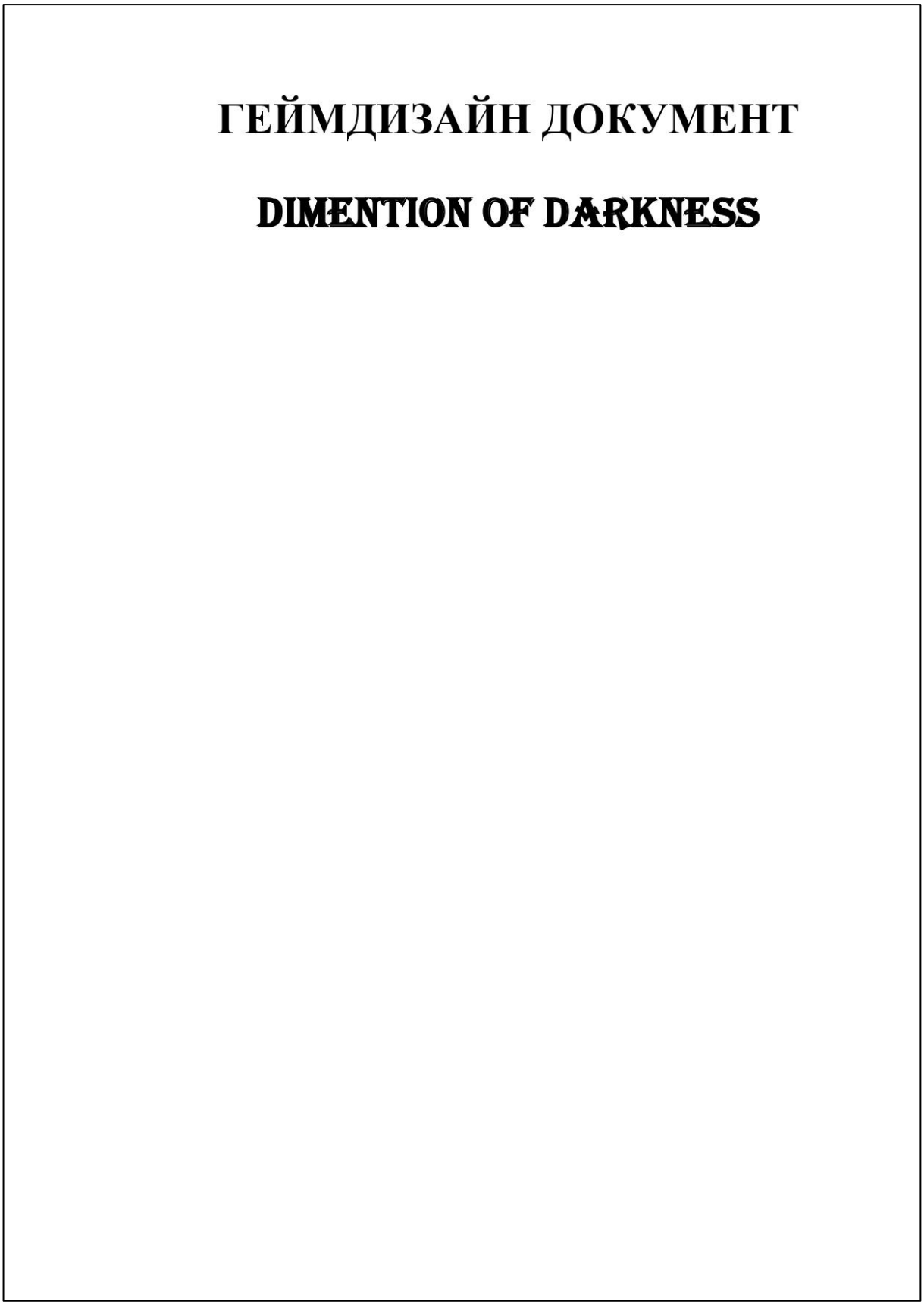


Рисунок В.1 – Сторінка 1 концепт-документу гри

1 СПЕЦИФІКАЦІЯ

1.1 Тетра

Механіка: Shoot`em-up

Технологія: PC

Історія: ГГ-науковець

Естетика: Fantastic + Darkness

1.2 Цільова аудіторія

Головною аудиторією нашої будуть чоловіки 16-40 років. Цю аудиторію мають привабити: динамічна система бою, графічні ефекти зброї, кооператив, можливість безмежно розвивати персонажа.

1.3 USP

- Спробуйте себе у ролі відчайдушного героя у невідомому темному вимірюванні, який намагається повернутись до дому;
- отримайте задоволення від динамічної бойової системи, знищуючи ворогів;
- не майте обмежень по рівню розвинення персонажа;
- безмежно знищуйте ворогів їдучих постійним натиском на вас.

1.4 Час ігрової сесії

Гра не має рівнів або місій тощо. Гра складається із забігів. Кожен забіг, на початковій стадії гри, може продовжуватись від 5 до 30 хвилин. З подальшим прогресом гравця середній час забігу буде збільшуватись.

Рисунок В.2 – Сторінка 2 концепт-документу гри

2 ГЕЙМПЛЕЙ

Головною метою гравця за гру буде назбирати достатню кількість енергії, за один забіг, щоб відремонтувати пристрій який допоможе повернутись нашому герою додому. Гравцю треба назбирати необхідну кількість енергії саме за один забіг, бо поміж забігами енергія накопичуватись не буде та при старті нового забігу стара енергія знищується. Це і є основне випробування для гравця. Також йому буде заважати збирати енергію постійний натиск потвор. Щоб мати змогу назбирати достатньо енергії за один раз, гравцю треба зробити багато забігів щоб вдосконалити зброю костюм тощо.

Гра ділиться на два етапи: забіг та відпочинок на базі. Забіг є основним етапом під час якого гравець повинен отримати основне задоволення від гри. Під час відпочинку на базі гравець має змогу: витратити зібрані ресурси на вдосконалення спорядження, відпочити та зберегти гру.

Рисунок В.3 – Сторінка 3 концепт-документу гри

3 ІСТОРІЯ ТА СЮЖЕТ

3.1 Вступ

Гра не є сюжетно орієнтованою, але гра має історію та мінімальний сюжет щоб гравець розумів логіку та причини того що відбувається під час гри.

3.2 Історія

2087 рік, вчені у науково-дослідницькому центрі досліджують паралельні вимірювання та можливості із ними взаємодіяти. Одного разу 32-річному вченому вдається винайти пристрій який може розривати простір та створювати стабільний портал із іншим вимірюванням, але він навіть не має здогадів що можна очікувати від того вимірювання. Цікавість бере верх над групою вчених та вони віршують розпочати експеримент по запуску зв'язуючого пристрою, не отримавши на це дозвіл від керівництва та не попередивши інших вчених.

Десяте листопада 2087 року, зв'язуючий пристрій під назвою PSDC (Possibly a Stable Dimensional Connector) не справляється з енергетичним навантаженням та бокові опори які тримали портал ламаються, портал одразу збільшується у розмірі та доходячи до розмірів усієї лабораторії, PSDC вимикається через перенавантаження, в наслідок чого лабораторія опиняється у іншому вимірювання. Дослідники мають деяке обладнання та генератор у лабораторії що дозволить їм прожити деякий час. Дослідникам треба відремонтувати PSDC тому вони вирішують відправитись за межі лабораторії для досліджень та вирішення проблеми.

Під час першого виходу з лабораторії, Уолтер, перший дослідник доброволець, бачить перед собою всюди тьму та густий також темний туман. Трохи озирнувшись Уолтер побачить дивне створіння схоже чи на павука чи на людину та воно вочевидь було налаштоване вороже. Зреагувавши Уолтер вбиває це створіння із пістолета, який йому видали. Трохи дослідивши довкілля, він розуміє, завдяки його спорядженню він

Рисунок В.4 – Сторінка 4 концепт-документу гри

може збирати з довкілля та створінь що тут живуть енергію, це допоможе вченим відремонтувати PSDC та повернутись до дому. Також потвори що тут живуть, як помітив Уолтер, створені із різних матеріалів що дозволяє використовувати їх залишки як матеріал для АНТ (atomic handheld transducer), це пристрій який є майже у кожного у 2087 році, він може розбирати речі на окремі атоми та створювати з них те що хоче власник пристрою.

Повернувшись до лабораторії Уолтер помічає що ємності його рюкзаку для збору енергії недостатньо щоб назбирати енергії за раз. Але що набагато гірше Уолтер розуміє темна енергія яку він назбирав дуже швидко анігілює з тою енергією що телепортувалась в це вимірювання. Через це вчені не мають змоги просто накопичувати енергію, значить їм потрібно вдосконалити рюкзак Уолтера щоб він міг принести багато енергії за раз.

Подальшою метою Уолтера є зібрати достатню кількість ресурсів щоб вдосконалити своє обладнання та відремонтувати PSDC щоб повернутись додому, але створіння що живуть у цьому вимірюванні не дадуть йому це так просто зробити.

Рисунок В.5 – Сторінка 5 концепт-документу гри

4 МЕХАНІКИ

4.1 Механіки гравця

Поява (у забігу)

Для того, щоб почати забіг, гравцеві (усім гравцям) необхідно зайти в телепорт. Після того почнеться сам забіг.

Поява (на базі)

Після того, як гравець у забігу загине, він опиниться на базі. Кількість твердотільного матеріалу скидається до 0.

Ривок та стрибок

Ривок дозволяє гравцеві практично моментально переміститися на певну дистанцію у напрямку руху гравця. Гравець може зробити ривок, натиснувши на Space. Ривок витрачатиме певну кількість витривалості.

Отримання твердотільного матеріалу та темної енергії з убитих супротивників

Після смерті противника, на рахунок гравця переходитиме певна кількість темної енергії, а також із противника може випасти якась кількість твердотільного матеріалу, з якого гравець зможе створити нові предмети.

Рисунок В.6 – Сторінка 6 концепт-документу гри

4.2 Механіки супротивника

Спавн

Вороги з'являтимуться навколо гравця, поза його полем зору. Частота появи противників залежить від часового інтервалу, а також від просування гравця по карті. Також будуть додаватися/мінятись типи противників, це залежатиме від просування гравця по карті та кількості накопичених одиниць темної енергії під час забігу.

Атака

Вороги зможуть використовувати різні види атак, такі як удари у ближньому бою, стрілянина зі зброї, кидання снарядів, магічні атаки тощо. Атаки деяких супротивників матимуть певний патерн. Залежно від типу ворога, буде змінюватися швидкість/збиток атаки. Також існуватимуть особливі атаки, наприклад атаки по області, уповільнення, зменшення огляду тощо.

Смерть

Смерть ворога супроводжуватиметься анімацією та/або візуальними ефектами. Після смерті противника на їхньому місці з'являтиметься ресурс, який гравець може підібрати та використати надалі. Також смерть противників може супроводжуватися певними особливими ефектами смерті, наприклад вибухом.

Рисунок В.7 – Сторінка 7 концепт-документу гри

4.3 Механіки світла та темряви

Механіка освітлення

Направляючи світло на туман, гравець буде розсіювати його, поки не перенаправить світло в іншу область. У освітленій області не можуть сповнитися супротивники. Для того, щоб висвітлити якусь область, існують певні предмети, що витрачаються, або поліпшення.

Механіка туману

У всіх місцях карти, які гравець ніяк не висвітлює, він бачитиме чорний туман. Навіть якщо гравець одного разу вже висвітлив область, то, переставши висвітлювати область, до неї повернеться туман.

4.4 Механіки зброї

Перезаряджання

Практично у всієї зброї в грі є свій час на перезарядку після пострілу, при цьому найчастіше зброя уповільнюватиме гравця під час перезарядки, а такі рухи як біг або стрибки через перешкоди збиватимуть таймер перезарядки, повертаючи його до початку відліку.

Заряд пострілу

Деяка зброя у грі завдає постріли після процесу заряду певної тривалості. Заряд здійснюється затисканням ЛКМ, після того, як гравець відпустить ЛКМ зброю зробить удар/постріл. Більшість зброї у грі з подібною механікою матиме максимальну межу заряду для удару/пострілу. Чим більшої потужності заряд, тим більшої сили буде постріл.

Рисунок В.8 – Сторінка 8 концепт-документу гри

Простріл противників

Деяка зброя у грі прострілюватиме супротивників наскрізь, наприклад, така зброя, як снайперська гвинтівка. При пострілі, куля з такої зброї пробиватиме всіх супротивників на своєму шляху, зупиняючись (зникаючи) на певній відстані.

4.5 Механіки костюму персонажа

Загальні відомості

Енергетичний щит

Костюм має вбудований енергетичний щит, він має певну кількість очок, які будуть забиратися при отриманні збитків від супротивників. Після того, як щит буде розряджений, гравець почне втрачати очки здоров'я від отриманої шкоди. Щит починає автоматично поступово заряджатися до максимального заряду поки гравець не отримує шкоди від супротивників певний час. Отримання втрат під час заряду щита зіб'є процес зарядки, кількість очок зберегтися на тому рівні, де зупинилася зарядка.

Рисунок В.9 – Сторінка 9 концепт-документу гри

5 ІНТЕРФЕС КОРИСТУВАЧА

5.1 HUD

На екрані гравець бачить:

- рівень свого здоров’я, витривалості та енергощита костюма;
- слоти зі зброєю та обрану зброю, а також максимальну та фактичну кількість патронів у магазині.;
- фактичну та необхідну для підвищення рівня гри кількість темної енергії.

5.2 Система контролю

Гравець має доступ до наступного контролю:

- Переміщення – [W][A][S][D];
- оглянутись навколо – миша;
- ривок/перестрибнути перешкоду – [Space];
- біг – [Shift];
- постріл/прицілювання – [LMB]/[RMB];
- особлива атака зброї – [Q];
- взаємодія із інтерактивними предметами – [E];
- меню паузи – [Esc].

5.3 Аудіо ефекти

Гра буде музичний супровід. Звукові ефекти від пострілів та заряду зброї, противники та персонаж також будуть мати свої звукові ефекти. А також деякі предмети навколо будуть створювати звукові ефекти.

Рисунок В.10 – Сторінка 10 концепт-документу гри

6 РЕФЕРЕНСИ

6.1 Головні герої

Усі виглядають приблизно однаково. Білий костюм-екзоскелет, зріст десь 1.8м

Рисунок В.11 – Сторінка 11 концепт-документу гри

Рисунок В.12 – Сторінка 12 концепт-документу гри

Рисунок В.13 – Сторінка 13 концепт-документу гри

6.2 Противники

Повністю чорні створіння, місяцями схожі на людей, місцями на тварин. Замість очей та роту чорні впадини.

Рисунок В.14 – Сторінка 14 концепт-документу гри

Рисунок В.15 – Сторінка 15 концепт-документу гри

Рисунок В.16 – Сторінка 16 концепт-документу гри

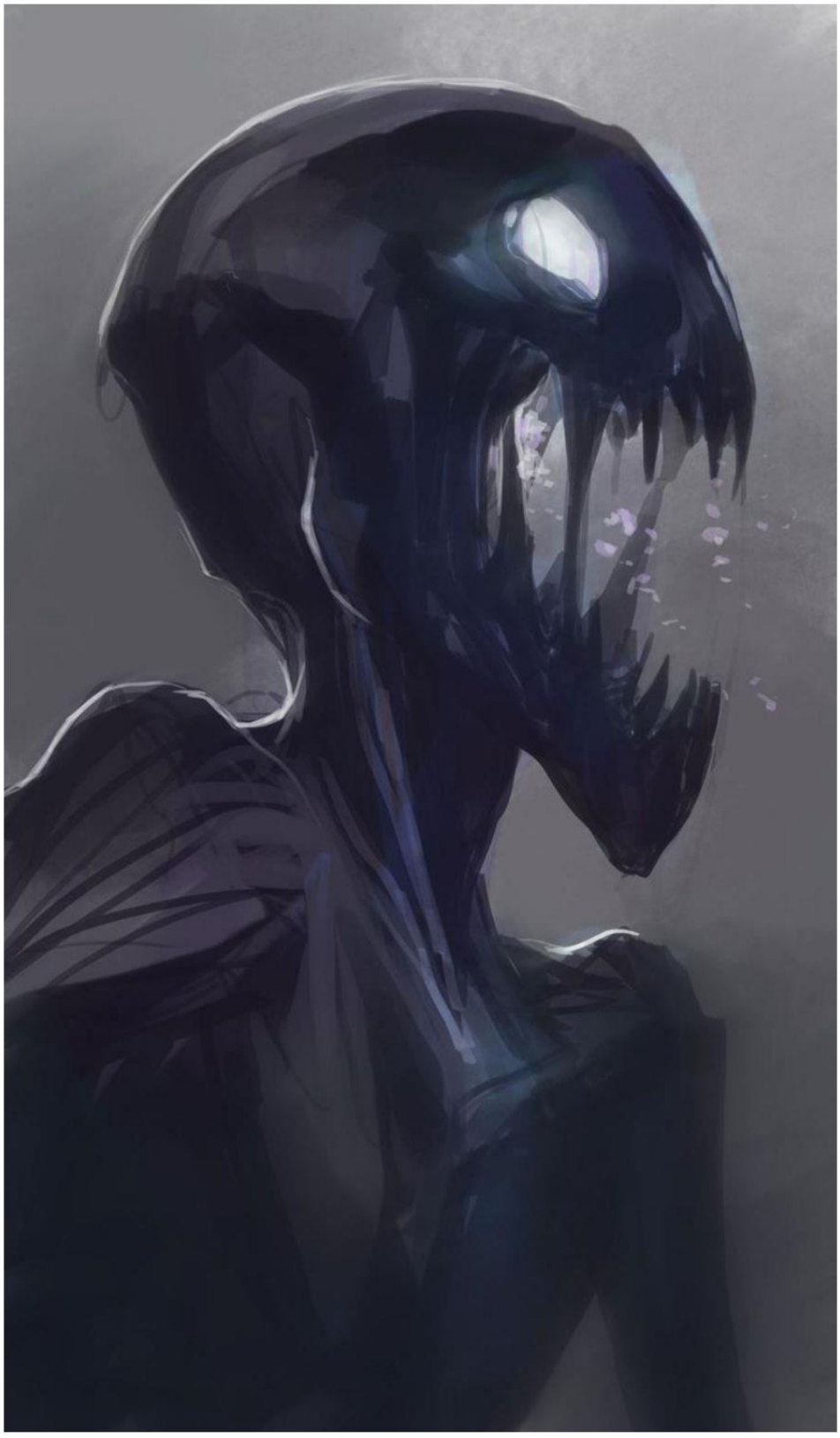


Рисунок В.17 – Сторінка 17 концепт-документу гри

6.3 Біом

Перший біом це звичайна земля рожево-коричневого кольору. По краям біомів будуть розломи, поламані мости, автомобілі та інше. Вид згори.

Рисунок В.18 – Сторінка 18 концепт-документу гри

Рисунок В.19 – Сторінка 19 концепт-документу гри

6.4 Лабораторія

Приміщення з великою кількістю поламаної техніки, такої як комп’ютери, прінтери, якісь колби і так далі. В лабораторії будуть місцями горстки землі, із полу стирчатимуть сталагміти. Задня частина лабораторії буде завалена та зливатиметься із землею. Тільки вид згори.

Рисунок В.20 – Сторінка 20 концепт-документу гри

Рисунок В.21 – Сторінка 21 концепт-документу гри

Рисунок В.22 – Сторінка 22 концепт-документу гри

6.5 Туман

Туман буде всюди, окрім шляху де світитиме ліхтар, або освітлювальні предмети.

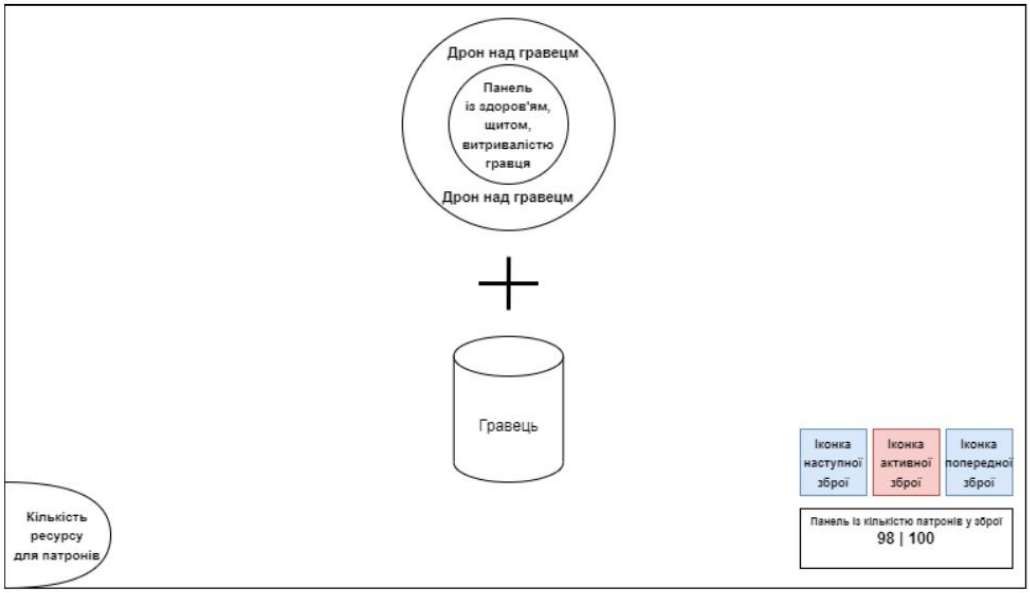
6.6 Головне меню

Головне меню буде складатися із фону, та кнопок «Одиночна гра», «Мультиплеєр», «Налаштування» та «Вихід».

Рисунок В.23 – Сторінка 23 концепт-документу гри

6.7 Меню налаштувань

6.8 Ігровий інтерфейс



7 РОЗГОРТАННЯ ТА РОЗРОБКА

7.1 Порядок встановлення

Рисунок В.24 – Сторінка 24 концепт-документу гри

Гру можливо буде інсталювати через інсталятор або через Steam/EGS

7.2 Системні вимоги

Операційна система: Windows 8 або кращє.

Процесор: Intel 2.77GHz Quad-core.

ОЗП: 8 Gb.

Відеокарта: NVIDIA GTX 550 Ti.

DirectX: 10.

Місце на диску: до 1Gb.

8 РОЗРОБКА

Гра буде розроблена на рушії Unity.

Репозиторій: https://github.com/Dayman267/ZombieSurviveSmash1.

Рисунок В.25 – Сторінка 25 концепт-документу гри

ДОДАТОК Г

Код генерації ландшафтів по тріггеру гравця

```
public void SpawnGameObjectsOnTriggerEnter(Vector3 position)
{
    float positionX = (position.x / objectSize) + width/2;
    float positionY = (position.z / objectSize) + height/2;
    int maxX = gameObjectsToSpawn.GetLength(0) - 1;
    int maxY = gameObjectsToSpawn.GetLength(1) - 1;
    if (positionX > 0 && gameObjectsToSpawn[(int)positionX - 1,
(int)positionY] == -1)
    {
        gameObjectsToSpawn[(int)positionX - 1,(int)positionY] =
Random.Range(0, pickedTerrains.Count);
        GameObject instance =
Instantiate(pickedTerrains[gameObjectsToSpawn[(int)positionX -
1,(int)positionY]],
            new Vector3((int)positionX - 1-width/2, 0,
(int)positionY-height/2) * objectSize,
            Quaternion.identity, parentToSet.transform);
        instance.SetActive(true);
        instance.GetComponent<Terrain>().enabled = true;
        instance.GetComponent<ObjectsGenerator>().enabled = true;
    }
    if (positionY > 0 && gameObjectsToSpawn[(int)positionX,
(int)positionY - 1] == -1)
    {
        gameObjectsToSpawn[(int)positionX, (int)positionY - 1] =
Random.Range(0, pickedTerrains.Count);
        GameObject instance =
Instantiate(pickedTerrains[gameObjectsToSpawn[(int)positionX,(int)pos
itionY - 1]],
            new Vector3((int)positionX -width/2, 0, (int)positionY -
1-height/2) * objectSize,
            Quaternion.identity, parentToSet.transform);
        instance.SetActive(true);
        instance.GetComponent<Terrain>().enabled = true;
        instance.GetComponent<ObjectsGenerator>().enabled = true;
    }
    if (positionX > 0 && positionY > 0 &&
gameObjectsToSpawn[(int)positionX - 1, (int)positionY - 1] == -1)
    {
        gameObjectsToSpawn[(int)positionX - 1, (int)positionY - 1] =
Random.Range(0, pickedTerrains.Count);
        GameObject instance =
Instantiate(pickedTerrains[gameObjectsToSpawn[(int)positionX -
1,(int)positionY -1]],
            new Vector3((int)positionX - 1-width/2, 0,
(int)positionY-1-height/2) * objectSize,
            Quaternion.identity, parentToSet.transform);
```

```
        instance.SetActive(true);
        instance.GetComponent<Terrain>().enabled = true;
        instance.GetComponent<ObjectsGenerator>().enabled = true;
    }
    if (positionX < maxX && gameObjectsToSpawn[(int)positionX + 1,
(int)positionY] == -1)
    {
        gameObjectsToSpawn[(int)positionX + 1, (int)positionY] =
Random.Range(0, pickedTerrains.Count);
        GameObject instance =
Instantiate(pickedTerrains[gameObjectsToSpawn[(int)positionX +
1,(int)positionY]],
            new Vector3((int)positionX + 1-width/2, 0,
(int)positionY-height/2) * objectSize,
            Quaternion.identity, parentToSet.transform);
        instance.SetActive(true);
        instance.GetComponent<Terrain>().enabled = true;
        instance.GetComponent<ObjectsGenerator>().enabled = true;
    }
    if (positionY < maxY && gameObjectsToSpawn[(int)positionX,
(int)positionY + 1] == -1)
    {
        gameObjectsToSpawn[(int)positionX, (int)positionY + 1] =
Random.Range(0, pickedTerrains.Count);
        GameObject instance =
Instantiate(pickedTerrains[gameObjectsToSpawn[(int)positionX,(int)pos
itionY+1]],
            new Vector3((int)positionX-width/2, 0, (int)positionY+1-
height/2) * objectSize,
            Quaternion.identity, parentToSet.transform);
        instance.SetActive(true);
        instance.GetComponent<Terrain>().enabled = true;
        instance.GetComponent<ObjectsGenerator>().enabled = true;
    }
    if (positionX < maxX && positionY < maxY &&
gameObjectsToSpawn[(int)positionX + 1, (int)positionY + 1] == -1)
    {
        gameObjectsToSpawn[(int)positionX + 1, (int)positionY + 1] =
Random.Range(0, pickedTerrains.Count);
        GameObject instance =
Instantiate(pickedTerrains[gameObjectsToSpawn[(int)positionX +
1,(int)positionY+1]],
            new Vector3((int)positionX + 1-width/2, 0,
(int)positionY+1-height/2) * objectSize,
            Quaternion.identity, parentToSet.transform);
        instance.SetActive(true);
        instance.GetComponent<Terrain>().enabled = true;
        instance.GetComponent<ObjectsGenerator>().enabled = true;
    }
    if (positionX > 0 && positionY < maxY &&
gameObjectsToSpawn[(int)positionX - 1, (int)positionY + 1] == -1)
```

```
    {
        gameObjectsToSpawn[(int)positionX - 1, (int)positionY + 1] =
Random.Range(0, pickedTerrains.Count);
        GameObject instance =
Instantiate(pickedTerrains[gameObjectsToSpawn[(int)positionX -
1,(int)positionY+1]],
            new Vector3((int)positionX - 1-width/2, 0,
(int)positionY+1-height/2) * objectSize,
            Quaternion.identity, parentToSet.transform);
        instance.SetActive(true);
        instance.GetComponent<Terrain>().enabled = true;
        instance.GetComponent<ObjectsGenerator>().enabled = true;
    }
    if (positionY > 0 && positionX < maxX &&
gameObjectsToSpawn[(int)positionX + 1, (int)positionY - 1] == -1)
    {
        gameObjectsToSpawn[(int)positionX + 1, (int)positionY - 1] =
Random.Range(0, pickedTerrains.Count);
        GameObject instance =
Instantiate(pickedTerrains[gameObjectsToSpawn[(int)positionX +
1,(int)positionY-1]],
            new Vector3((int)positionX + 1-width/2, 0,
(int)positionY-1-height/2) * objectSize,
            Quaternion.identity, parentToSet.transform);
        instance.SetActive(true);
        instance.GetComponent<Terrain>().enabled = true;
        instance.GetComponent<ObjectsGenerator>().enabled = true;
    }
}
```