Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»

РОЗРОБКА 2D ADVENTURE RPG «LAST DAY ON THE TRAIN» З ГЕНЕРАЦІЄЮ РІВНІВ В ПІКСЕЛЬНОМУ СТИЛІ З ВИКОРИСТАННЯМ UNITY

DEVELOPMENT OF THE 2D ADVENTURE RPG «LAST DAY ON THE TRAIN» WITH PIXEL-STYLE LEVEL GENERATION USING UNITY

спеціальність 121 «Інженерія програмного забезпечення»
освітня програма «Інженерія програмного забезпечення»

Виконав: здобувач вищої освіти
групи ІПЗ-41
Данилко Костянтин Сергійович

Керівник:
Вознюк Анастасія Вадимівна

Кваліфікаційну роботу
допущено до захисту
«10» 06 2025 р.
Гарант освітньої програми:
к.т.н., доцент
Ліщина Наталія Миколаївна

Луцьк – 2025 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *інженерії програмного забезпечення*
Ступінь вищої освіти *бакалавр*
Галузь знань: *12 «Інформаційні технології»*
Спеціальність: *121 «Інженерія програмного забезпечення»*
Освітня програма: «*Інженерія програмного забезпечення»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____
«20» 12 2024 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Данилку Костянтину Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи *«Розробка 2D adventure RPG «Last day on the train» з генерацією рівнів в піксельному стилі з використанням Unity»*
Керівник роботи: *асистент Вознюк А. В*
3. Вихідні дані до роботи: *Unity, C#, Aseprite, Git, Trello, методичні вказівки до виконання кваліфікаційної роботи бакалавра*
4. Зміст розрахунково-пояснювальної записки (перелік питань, що потрібно розробити): *сформувати мету та завдання кваліфікаційної роботи, визначити вимоги до розроблюваної гри, вибрати засоби та технології розробки. Розробити ігрові механіки, створити графічний контент та просести тестування*
5. Перелік графічного матеріалу: *20 рисунків, 1 додаток*

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз предметної області	*Вознюк А. В.*		
Специфікація вимог до розробленої системи	*Вознюк А. В.*		
Розробка об`єкта проектування	*Вознюк А. В.*		
Нормоконтроль	*Вознюк А. В.*		
Гарант ОП	*Ліщина Н. М.*		
Показник запозичень тексту	0.86 %		
Академічна доброчесність	*Вознюк А. В.*		

7. Дата видачі завдання *«20» грудня 2024 р.*

№	Назва етапів кваліфікаційної роботи бакалавра	Строк виконання етапів роботи	Примітка
1	Огляд літературних джерел по темі кваліфікаційної роботи бакалавра	до 04.02.2025 р.	Виконано
2	Аналіз проблеми розробки та впровадження об`єкту проектування	до 01.03.2025 р.	Виконано
3	Обґрунтування вибору шляхів, технологій і засобів вирішення поставленого завдання	до 15.03.2025 р.	Виконано
4	Розробка функціонально-структурної схеми роботи об'єкта проектування та проектування бази даних	до 29.03.2025 р.	Виконано
5	Практична реалізація об'єкта проектування та розробка бази даних	до 26.04.2025 р.	Виконано
6	Тестування та налагодження об'єкта проектування	до 03.05.2025 р.	Виконано
7	Здача чистового варіанту кваліфікаційної роботи бакалавра на кафедру	до 10.06.2025 р.	Виконано

Здобувач вищої освіти _____ Костянтин ДАНИЛКО

Керівник кваліфікаційної роботи _____ Анастасія ВОЗНЮК

АНОТАЦІЯ

Данилко К. С. Розробка 2D Adventure RPG в піксельному стилі з генерацією рівнів з використанням Unity. Рукопис. Кваліфікаційна робота бакалавра ОП «Інженерія програмного забезпечення» спеціальності «Інженерія програмного забезпечення». Луцький національний технічний університет. Луцьк, 2025.

Кваліфікаційна робота бакалавра складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків.

У першому розділі здійснено огляд сучасних методів генерації ігрових рівнів, аналіз рушіїв для створення 2D ігор та графічних редакторів для створення піксельної графіки, а також сформульовано завдання на кваліфікаційну роботу. У другому розділі сформовано функціональні та нефункціональні вимоги до гри та обґрунтовано вибір засобів реалізації, зокрема обрано ігровий рушій та редактор графіки. У третьому розділі описано процес розробки та тестування прототипу гри, реалізацію генерації рівнів, основну механіку взаємодії, розміщення об’єктів оточення та загальне тестування працездатності. У висновках узагальнено результати виконання роботи, досягнення поставлених цілей та визначено перспективи подальшого розвитку.

Ключові слова: Unity, піксельна графіка, 2D RPG, процедурна генерація, шум Перліна, розробка ігор.

ABSTRACT

Danylko K. Development of a 2D adventure RPG in pixel art style with level generation using Unity. Manuscript. Qualification work of the bachelor’s program "Software engineering" in the specialty "Software engineering". Lutsk National Technical University. Lutsk, 2025.

The bachelor’s qualification thesis consists of an introduction, three chapters, conclusions, a list of references, and appendices.

The first chapter presents an overview of modern procedural generation methods, an analysis of existing game development solutions, as well as graphic editors for creating pixel art, and formulates the project’s objectives. The second chapter defines the functional and non-functional requirements for the project and justifies the selection of development tools and technologies, including game engine and program for graphical content. The third chapter describes the development process and testing of the game prototype, implementation of level generation, interaction mechanics, environment placement, and overall performance testing. The conclusions summarize the results of the project, the degree to which the objectives were met, and outline directions for further improvement.

Keywords: Unity, pixel art, 2D RPG, procedural generation, Perlin noise, game development, software engineering.

ЗМІСТ

ВСТУП

Актуальність теми. Сучасні технології розробки відеоігор дозволяють створювати складні та динамічні ігрові світи, що забезпечують унікальний досвід для гравців. Одним із важливих аспектів цього процесу є процедурна генерація рівнів, яка дозволяє зробити ігровий процес більш варіативним, підвищуючи реіграбельність. Завдяки цьому підходу кожне проходження гри може суттєво відрізнятися від попереднього, що робить її цікавішою для користувачів.

Світові тенденції у сфері геймдизайну демонструють зростаючу популярність ігор із процедурно згенерованими рівнями. Такі проєкти, як Dead Cells, Hades, Enter the Gungeon, показують, що правильний підхід до генерації рівнів може зробити гру успішною та популярною серед гравців. Однак багато з цих ігор мають власні обмеження, тому розробка нової RPG з акцентом на процедурну генерацію та адаптивний ігровий процес є перспективним напрямом.

Попри широкий вибір сучасних ігор із процедурною генерацією, багато з них не враховують балансу між випадковістю та структурованістю рівнів, що може негативно впливати на якість ігрового процесу. Деякі алгоритми забезпечують надмірну хаотичність рівнів, тоді як інші обмежують можливості варіативності. Тому актуальним є розроблення гри, яка гармонійно поєднуватиме алгоритми випадкової генерації з контрольованими механізмами формування ігрового світу.

Метою цієї кваліфікаційної роботи є створення 2D гри у піксельному стилі з елементами пригодницької RPG, що базується на сучасних підходах до процедурної генерації рівнів.

Об’єктом кваліфікаційної роботи бакалавра є процес розробки двовимірних ігор з процедурною генерацією рівнів.

Предметом кваліфікаційної роботи виступають інструменти, методи та алгоритми створення процедурного контенту у 2D іграх, а також особливості реалізації подібних рішень у середовищі Unity.

Для досягнення мети необхідно виконати такі завдання:

– провести аналіз предметної області, а саме наявні алгоритми процедурної генерації, актуальність піксельної графіки в сучасному геймдеві та популярності 2D ігор у піксельному стилі;

– сформулювати функціональні та нефункціональні вимоги до розроблюваної гри, побудувати UML-діаграму варіантів використання для демонстрації основних сценаріїв взаємодії гравця з системою;

– обрати засоби реалізації проєкту, а саме основний рушій розробки гри, алгоритм генерації рівнів, AI для керування поведінкою ворогів, графічний редактор, а також систему контролю версій та інструмент для організації робочого процесу;

– реалізувати прототип гри який відповідає поставленим вимогам;

– провести тестування функціональності та стабільності роботи гри, виконати налагодження та оптимізацію програмного коду;

– розробити інтерфейс та графічний контент для гри;

– підготувати документацію до розробленого програмного продукту;

– створити інсталяційний пакет для розгортання гри користувачем.

Отже, ця робота спрямована на створення якісної гри, яка поєднуватиме цікаві геймплейні механіки, процедурну генерацію рівнів та збалансований ігровий процес, що забезпечить користувачам унікальний досвід у світі піксельної RPG.

Робота апробована шляхом участі у X Міжнародній науково-практичній конференції з проблем вищої освіти і науки «Інформаційні технології в освіті, науці і виробництві (ІТОНВ-2025)», тези доповідей подані в додатку А.

РОЗДІЛ 1
АНАЛІЗ РОЗРОБКИ 2D ІГОР З ГЕНЕРАЦІЄЮ РІВНІВ ТА ПОСТАНОВКА ЗАВДАНЬ НА КВАЛІФІКАЦІЙНУ РОБОТУ

1.1 Аналіз сучасного стану проблеми

Жанр 2D Adventure RPG є популярним серед гравців завдяки можливості дослідження світу, розвитку персонажа та нелінійному ігровому процесу. Однією з важливих складових таких ігор є механіка генерації рівнів, яка може бути як статичною (заздалегідь створені карти), так і процедурною (автоматично згенеровані локації).

Процедурна генерація рівнів відіграє важливу роль у створенні унікального ігрового досвіду, забезпечуючи варіативність кожного проходження (рис. 1.1). Це особливо актуально для жанрів roguelike та roguelite, де головний акцент робиться на багаторазовому проходженні гри. Завдяки процедурному підходу рівні кожного разу виглядають інакше, що дозволяє гравцеві відчувати новизну та випробовувати різні стратегії замість запам’ятовування наперед заданих карт. Це підвищує реіграбельність, робить геймплей динамічнішим і запобігає швидкому насиченню контентом.

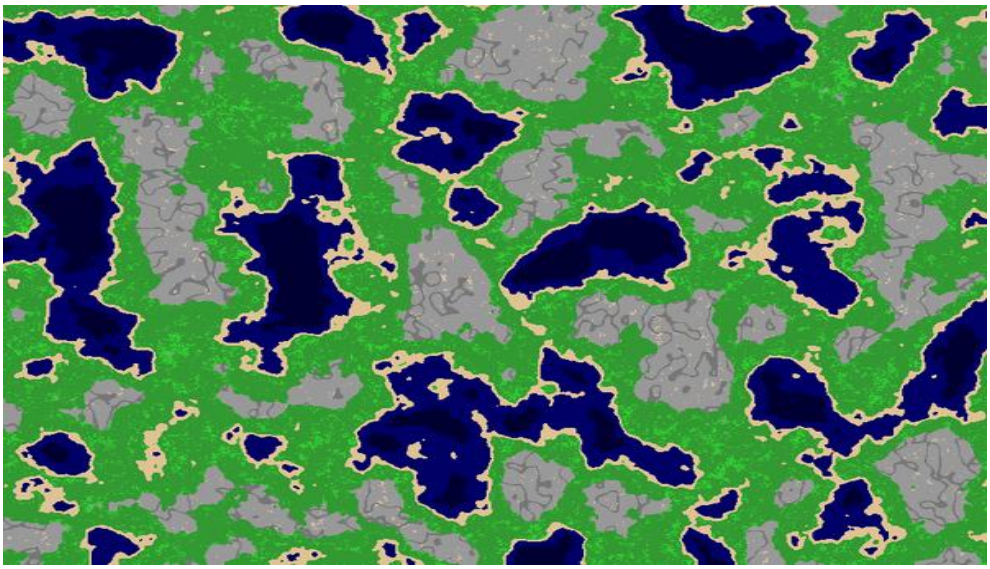

Рисунок 1.1 – Приклад генерації [10]

У сучасній розробці ігор важливо поєднувати процедурну генерацію з продуманим дизайном, щоб забезпечити баланс між випадковістю та контрольованою структурою рівнів. Повна випадковість може зробити ігровий процес хаотичним і незбалансованим, тому найчастіше використовують комбінацію алгоритмічної генерації та заздалегідь підготовлених елементів. Наприклад, у багатьох іграх застосовуються модульні рівні, де випадковим чином поєднуються заздалегідь створені сегменти, або використовується генерація на основі правил, що гарантує коректне розміщення об’єктів і логічність ігрового середовища.

Впровадження процедурної генерації також впливає на баланс гри, оскільки змінні умови проходження можуть ускладнювати чи спрощувати завдання для гравця.

Тому розробники часто додають механізми адаптації складності, які аналізують прогрес гравця та змінюють параметри генерації відповідно до його навичок. Це може включати регулювання кількості ворогів, розміщення ресурсів або динамічне коригування структури рівнів.

Процедурна генерація є потужним інструментом, що дозволяє значно урізноманітнити ігровий процес і зробити кожне проходження унікальним. Вона не тільки спрощує створення великої кількості контенту, але й допомагає підтримувати інтерес гравця, стимулюючи його до повторного проходження та пошуку нових шляхів вирішення ігрових завдань.

Існує багато різних алгоритмів генерації карти та рівнів, у кожного є свої переваги та недоліки і використовувати їх потрібно залежно від того, на що ми хочемо зробити акцент та який тип гри оберемо, для початку потрібно більш детальніше дослідити найпопулярніші з них.

Алгоритм клітинних автоматів є одним із найефективніших методів для створення печероподібних рівнів. Він працює шляхом ітераційного оновлення тайлової сітки на основі певних правил, що визначають, які клітинки залишаться твердими, а які стануть прохідними. Завдяки цьому підходу можна отримати природні та нерівномірні форми печер або підземель, що робить його ідеальним

для жанру roguelike. Однією з головних переваг цього алгоритму є його здатність створювати зв’язні ігрові простори без необхідності ручного налаштування кожного рівня.

Алгоритм хвильового колапсу відзначається тим, що він дозволяє створювати узгоджені структури рівня на основі набору правил. Він працює за принципом розгортання тайлів у певному порядку, враховуючи обмеження на їхні можливі сусіди. Завдяки цьому підходу генерація рівня може дотримуватися дизайнерських принципів, що робить його корисним для створення більш структурованих та естетично узгоджених карт. Хвильовий колапс часто використовується у випадках, коли необхідно поєднати випадковість із певним рівнем контролю над формою рівня.

Графові алгоритми застосовуються для створення рівнів із логічною структурою, наприклад, розташування кімнат і коридорів. Вони дозволяють контролювати топологію рівня, визначаючи, як різні ділянки пов’язані між собою. Наприклад, у грі з підземеллями можна спочатку згенерувати граф із вершинами (кімнатами) і ребрами (проходами), а потім трансформувати цей граф у тайлову сітку. Такий підхід забезпечує продумане розташування ключових ігрових елементів, таких як точки входу та виходу, пастки, скарби або вороги.

Ще одним важливим методом процедурної генерації є використання шумових функцій, таких як шум Перліна (рис. 1.2) або просторовий шум Ворлі. Вони широко застосовуються для створення плавних ландшафтів, висотних карт, біомів або текстур. Шум Перліна дозволяє генерувати поступові зміни у висоті або текстурі, що робить його особливо корисним для створення природних середовищ [5]. Його можна використовувати для визначення типів місцевості, наприклад, розташування води, лісів чи гір. Шум Ворлі, у свою чергу, добре підходить для генерації клітиноподібних структур, наприклад, розподілу островів або печер.

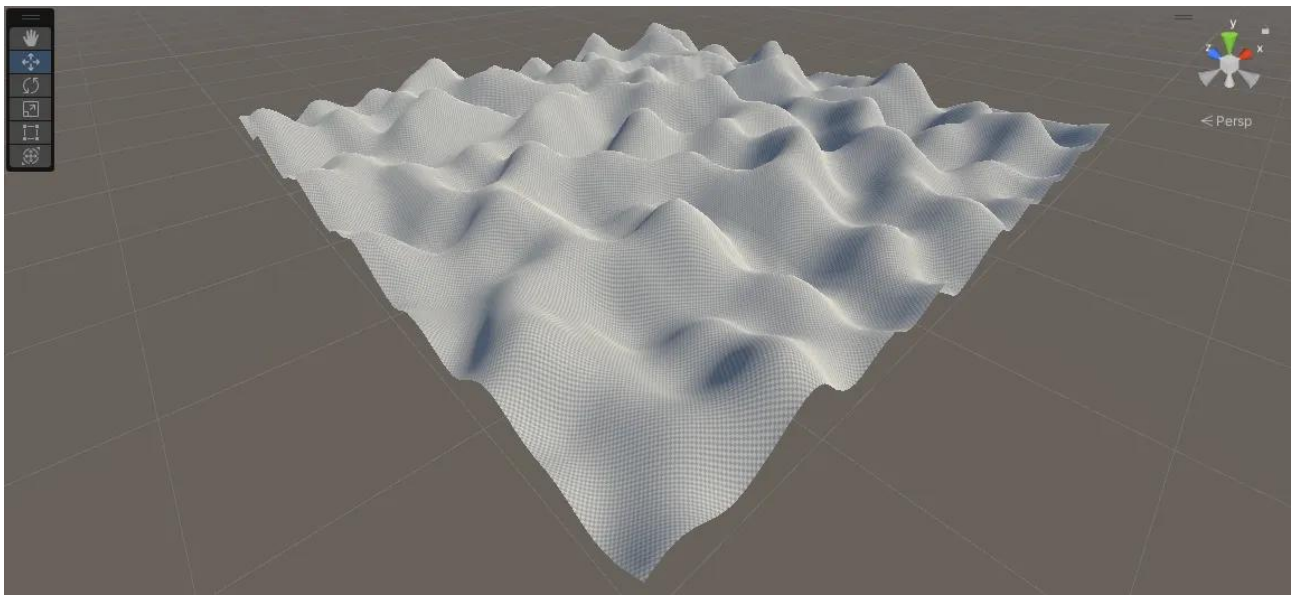

Рисунок 1.2 – Шум Перліна [16]

Зважаючи на огляд наявних рішень, доцільно обрати комбінований підхід, який поєднує тайлову генерацію, та шумові функції для забезпечення як випадковості, так і контрольованої структури рівнів. У такій моделі графові алгоритми можуть визначати загальну структуру рівня (наприклад, розташування основних кімнат і проходів), алгоритм клітинних автоматів може надавати рівню органічніші форми, а шумові функції можуть відповідати за деталізацію, додаючи варіативність у розподіл об’єктів і ландшафту.

Під час розробки гри постає важливе питання вибору графічного стилю, адже саме візуальна складова формує перше враження користувача. Серед найпоширеніших стилів сьогодні можна виділити 3D-графіку, векторну 2D, а також піксель-арт. Кожен із цих підходів має свої переваги. 3D дає більше можливостей у візуальному плані, але потребує значних ресурсів і часу на розробку, векторна 2D-графіка виглядає чисто й акуратно, проте іноді може втратити атмосферність, натомість піксельна графіка, хоч і стилізована під старі ігри, дозволяє створити унікальний візуальний вигляд при значно нижчих витратах ресурсів.

Піксель-арт активно використовується як інді-розробниками, так і більшими студіями, які прагнуть створити атмосферну і водночас технологічно простішу гру. З технічної точки зору, піксельна графіка має менші вимоги до

продуктивності, що дає змогу запускати гру на ширшому спектрі пристроїв, зокрема мобільних, а також дозволяє швидше змінювати або додавати новий контент без необхідності довготривалого рендерингу моделей або анімацій.

Останніми роками піксельні ігри залишаються актуальними та часто потрапляють у топи популярних тайтлів (рис. 1.3). Прикладами можуть бути такі хіти, як Stardew Valley, Celeste, Terraria, Dead Cells та інші. Вони поєднують в собі сучасні ігрові механіки з ностальгійною візуальною стилістикою, що дуже добре сприймається широкою аудиторією.

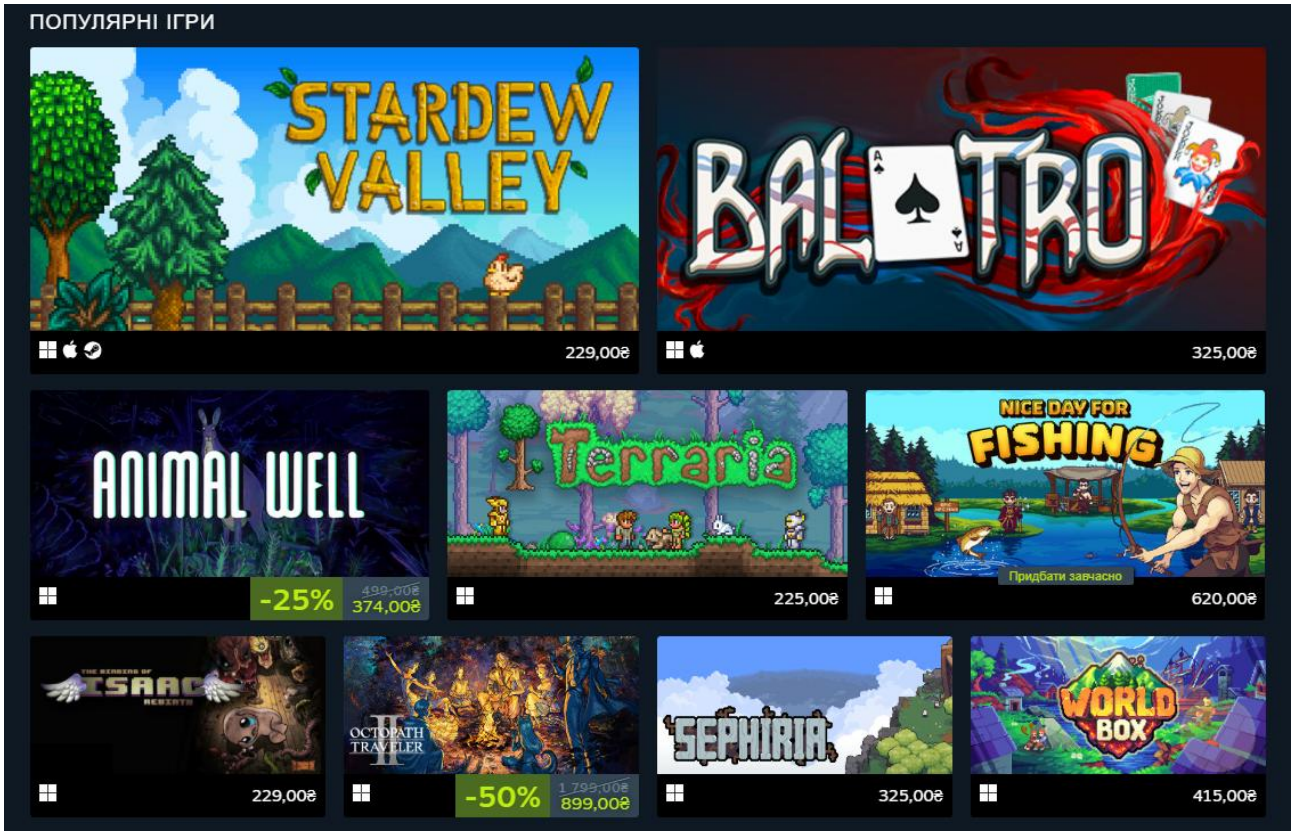

Рисунок 1.3 – Сторінка популярних ігор у Steam [13]

Окрім естетики, піксельна графіка дозволяє зосередитися на геймплеї та механіках, оминаючи зайві витрати на візуальні ефекти. Це дає змогу меншим командам чи соло-розробникам реалізовувати повноцінні ігрові проєкти в коротші строки та з меншими ресурсами. Для гри, в якій важлива механіка, процедурна генерація, взаємодія з об’єктами, вибір піксельного стилю є цілком виправданим.

В результаті вибір піксельної графіки ґрунтується не лише на технічній доцільності, а й на сучасній тенденції інтересу до подібної стилістики. Вона дозволяє створити привабливу та атмосферну гру, зберігаючи оптимальний баланс між якістю, продуктивністю та швидкістю розробки.

Вибір жанру є одним із ключових етапів на початку розробки гри, адже саме жанр визначає основні механіки, структуру геймплею та цільову аудиторію. У межах даної кваліфікаційної роботи було прийнято рішення розробляти гру в жанрі 2D adventure RPG, що поєднує в собі елементи пригодницької гри, рольової системи та бойових механік.

Один із основних факторів, що вплинув на вибір цього жанру, – його популярність серед інді-розробників та гравців, які цінують поєднання волі в діях з розвитком персонажа. Багато сучасних ігор, підтверджують актуальність жанру та можливість реалізувати в ньому цікаві механіки навіть у межах невеликої команди чи одиночної розробки. Жанр adventure RPG надає гнучкість у створенні як бойових, так і сюжетних компонентів.

Крім того, цей жанр ідеально поєднується з піксельною 2D графікою, яка є менш ресурсоємною та дозволяє зосередитися на геймплеї й дизайні рівнів. Використання процедурної генерації у межах RPG відкриває додаткові можливості щоб продовжувати грати або почати з нуля. Таким чином, вибір жанру adventure RPG є оптимальним як з технічного, так і з креативного погляду.

Розглянемо кілька відомих 2D RPG, які використовують різні методи генерації рівнів.

У The Binding of Isaac процедурна генерація лежить в основі ігрового процесу [14]. Під час кожного проходження гра формує карту з кімнат, які з’єднані між собою у випадковій послідовності (рис. 1.4). Основні елементи, такі як стартова кімната, шлях до кімнати з босом, додаткові приміщення генеруються за допомогою шаблонного підходу з використанням псевдовипадкових значень. Це дозволяє досягти варіативності та зберігати інтерес до гри протягом десятків годин.

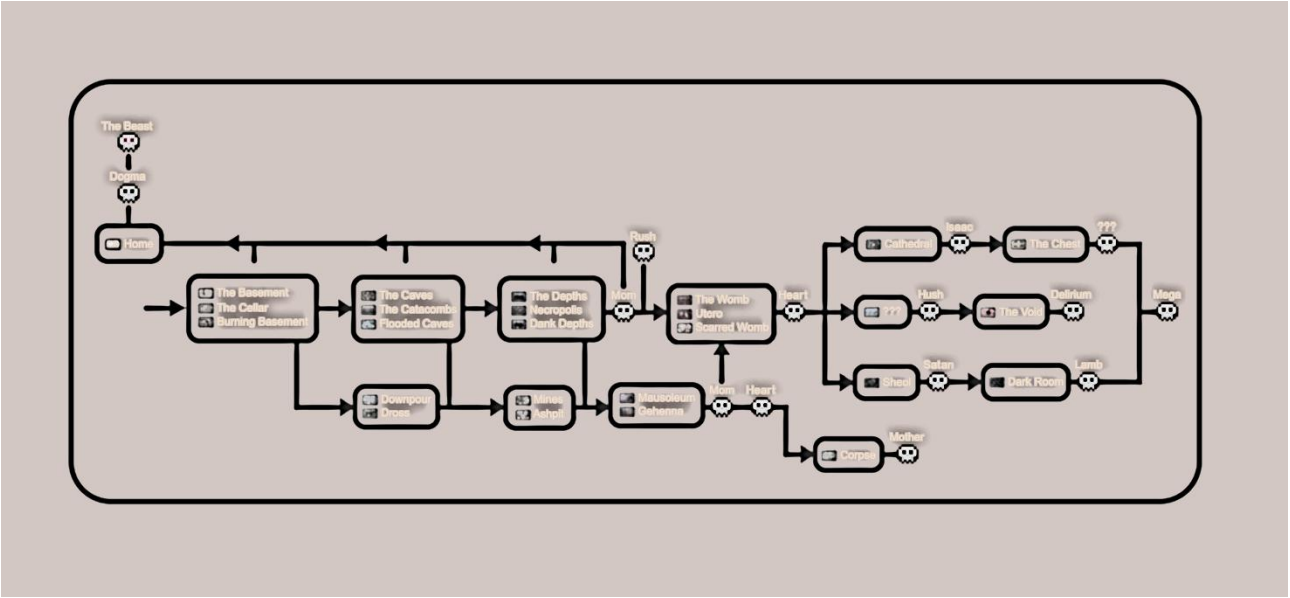

Рисунок 1.4 – Схема згенерованих рівнів у грі The Binding of Isaac [14]

Кожна кімната наповнюється динамічно: вороги, перешкоди, предмети й бонуси з’являються відповідно до заданих правил складності. Генерація враховує поточний рівень, прогрес гравця і збережений баланс. Крім структури рівнів, система процедурно вибирає з великої кількості предметів, які можуть з’явитися, забезпечуючи унікальні комбінації підсилень щоразу.

Для створення природного розподілу елементів іноді застосовується шум Перліна – алгоритм, який генерує плавні випадкові значення, що дозволяє уникати штучності та повторюваності візуального оформлення або розміщення об’єктів. Його часто використовують у мікрогенерації – наприклад, у варіації фону, деталей чи органічних структур.

Основним недоліком такого підходу є те, що при великій кількості проходжень гравець все одно починає помічати повторювані шаблони. Незважаючи на випадковість, обмежений набір кімнат, ворогів і подій може призводити до відчуття одноманітності. Також можливі ситуації, коли випадкова генерація створює надто складні або надто легкі рівні, що впливає на баланс гри.

У Dead Cells процедурна генерація використовується для створення рівнів, що поєднують елементи заготовлених шаблонів із випадковістю (рис. 1.5). Основна структура кожної локації визначається заздалегідь: розміри, типи зон,

наявність вертикальних і горизонтальних проходів. Проте при кожному новому проходженні ці елементи змішуються та з’єднуються по-різному, що створює нову конфігурацію ігрового простору. Завдяки цьому кожна сесія відчувається унікальним і спонукає гравця досліджувати карту.

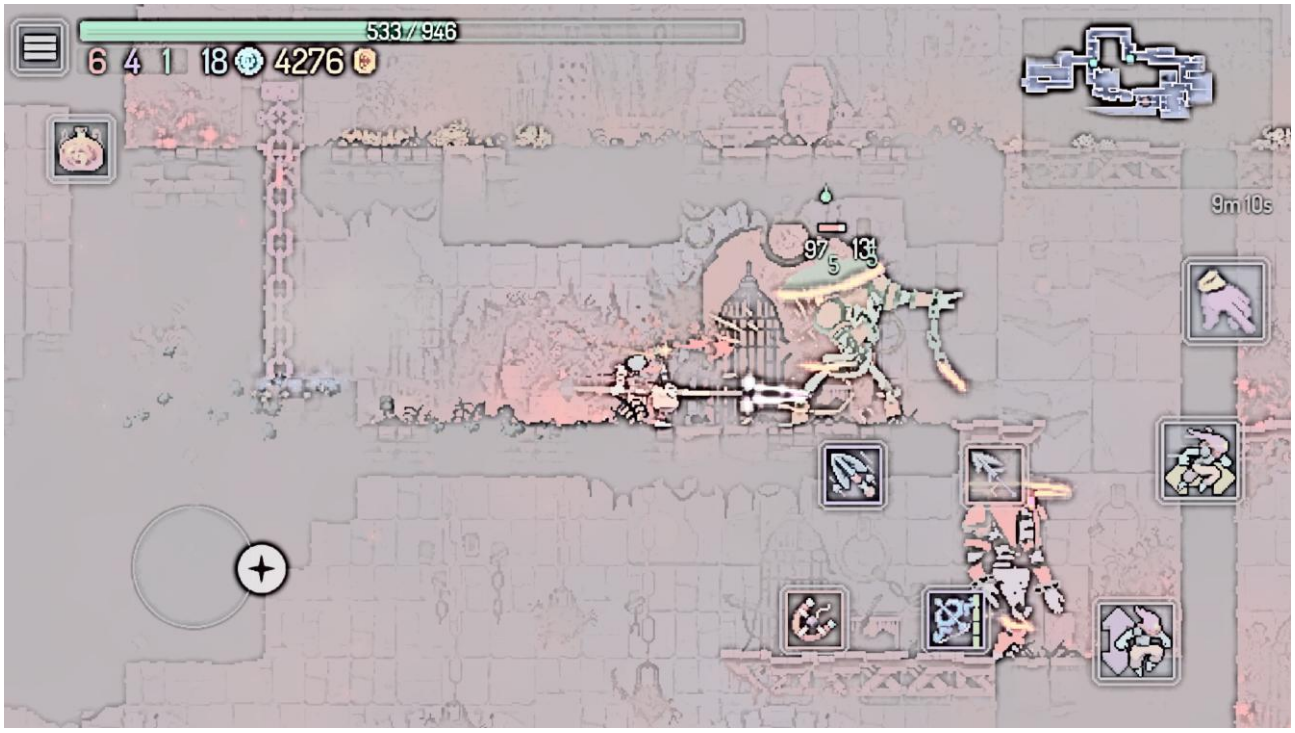

Рисунок 1.5 – Dead Cells [7]

Розміщення ворогів, пасток, скринь, телепортів, секретів та інших об’єктів також відбувається на основі випадковості, з урахуванням складності, яка прогресує разом із гравцем. Бойова динаміка у Dead Cells вимагає швидких рішень, тому локації мають бути водночас несподіваними, але й добре прорахованими з точки зору навігації. Для цього розробники використовують блоки кімнат і алгоритми, які враховують гравітацію, напрям руху та очікувану траєкторію гравця.

Шум Перліна або подібні алгоритми використовуються не лише для розміщення декоративних об’єктів, а й для формування плавних переходів між ярусами, варіації візуального оформлення зон і природного вигляду деяких

елементів навколишнього середовища. Це дозволяє уникнути візуальної монотонності та надати кожному рівню свій характер.

Недоліком такої генерації є потенційна втрата чіткого дизайнерського контролю. Хоча структура рівнів добре продумана, іноді генеруються надто порожні або перевантажені ділянки. Це може впливати на ритм гри, знижуючи динаміку або, навпаки, створюючи ситуації, коли гравець не має достатньо простору для маневру.

У Spelunky процедурна генерація є центральною механікою, яка формує основні рівні кожного проходження (рис. 1.6). Гра ділить простір на блоки, які розміщуються за допомогою алгоритмів випадковості з певними обмеженнями. Кожен рівень створюється на основі сітки – наприклад, кімнати розміром 4 на 4, де кожна з них має внутрішню структуру, створену або за шаблоном, або з використанням набору заздалегідь підготовлених фрагментів. Завдяки цьому, хоч рівні завжди нові, вони зберігають логічну ігрову структуру.

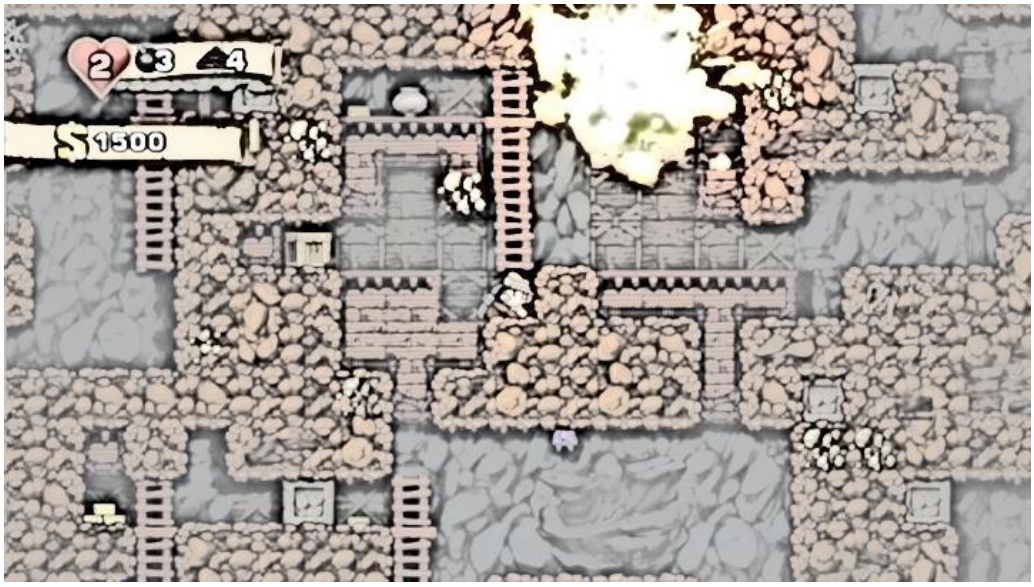

Рисунок 1.6 – Spelunky [12]

Генерація враховує розміщення виходу, пасток, ворогів, скарбів та секретів. Важливим аспектом є вертикальність – гравець завжди рухається згори вниз, тож алгоритми формують такі проходи, які забезпечують прохідність та підтримують баланс між ризиком і нагородою. Багато елементів гри – наприклад,

камені, лавові озера чи скелети – генеруються з урахуванням ймовірності, але в межах заданого діапазону, що дозволяє створити ефект постійної небезпеки та необхідності приймати рішення «на ходу».

Для візуального урізноманітнення та плавного розподілу об’єктів може використовуватись шум Перліна або інші генератори шуму. Вони допомагають уникнути надто «кубічного» вигляду карти та додають природності розташуванню блоків та декорацій.

Слабке місце такої генерації полягає в можливості появи непередбачуваних або несправедливих ситуацій – наприклад, коли пастка розміщується одразу біля точки старту, або вихід блокується складним набором перешкод. Це може зробити гру надто важкою для гравців, оскільки навіть при дотриманні загальних правил генерації, алгоритм не завжди гарантує ідеальний баланс.

Отже, у межах цієї кваліфікаційної роботи буде розроблена 2D гра у піксельному стилі, що використовує комбіновану генерацію рівнів, орієнтовану на створення цікавого, варіативного та збалансованого ігрового досвіду. Завдяки поєднанню різних підходів можна забезпечити як випадковість і унікальність кожного проходження, так і певний рівень контролю над логікою ігрового світу. Це дозволить створити динамічне ігрове середовище, що підлаштовується під гравця та підтримує його інтерес до дослідження процедурно згенерованих просторів.

1.2 Постановка завдання на кваліфікаційну роботу бакалавра

Для досягнення поставленої мети створити 2D гру в піксельному стилі з генерацією рівнів необхідно поставити чіткі завдання для виконання даної кваліфікаційної роботи, а саме:

– провести аналіз предметної області, а саме наявні алгоритми процедурної генерації, актуальність піксельної графіки в сучасному геймдеві та популярності 2D ігор у піксельному стилі;

– сформулювати функціональні та нефункціональні вимоги до розроблюваної гри, побудувати UML-діаграму варіантів використання для демонстрації основних сценаріїв взаємодії гравця з системою;

– обрати засоби реалізації проєкту, а саме основний рушій розробки гри, алгоритм генерації рівнів, AI для керування поведінкою ворогів, графічний редактор, а також систему контролю версій та інструмент для організації робочого процесу;

– реалізувати прототип гри який відповідає поставленим вимогам;

– провести тестування функціональності та стабільності роботи гри, виконати налагодження та оптимізацію програмного коду;

– розробити інтерфейс та графічний контент для гри;

– підготувати документацію до розробленого програмного продукту;

– створити інсталяційний пакет для розгортання гри користувачем.

РОЗДІЛ 2
СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ

2.1 Аналіз, визначення вимог до розроблюваного програмного забезпечення та проектування програмного забезпечення

В результаті аналізу проведеного в розділі 1.1 було визначено основні функціональні та нефункціональні вимоги до розроблюваної гри, а також сформовано базову архітектуру та логіку роботи системи. Основна мета – створення 2D гри з процедурною генерацією рівнів, що забезпечує варіативний ігровий процес, систему прогресу персонажа та збалансовану бойову механіку.

Для ефективного проєктування ігрової логіки важливо чітко визначити дії, які користувач зможе виконувати під час гри. Це дозволяє не лише краще організувати архітектуру програмного продукту, але й забезпечити зручний та зрозумілий інтерфейс.

З цією метою була побудована діаграма варіантів використання, яка зображена на рисунку 2.1.

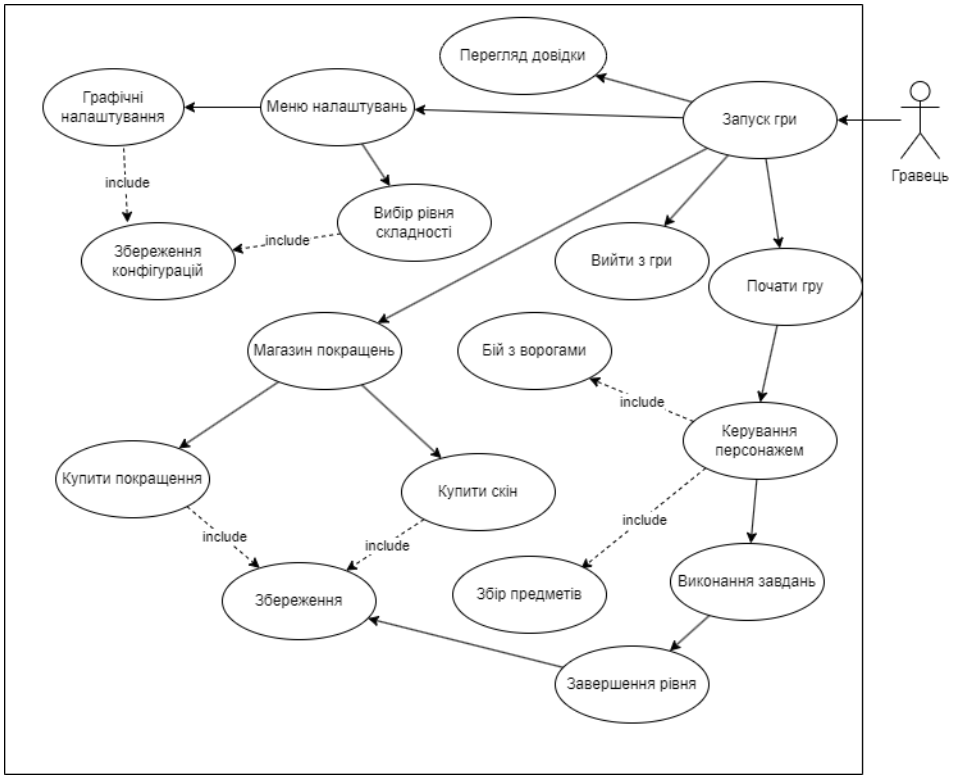

Рисунок 2.1 – Діаграма варіантів використання

Діаграма демонструє основні сценарії взаємодії гравця з системою. У ній відображено ключові функції гри – від запуску й управління персонажем до зміни налаштувань і збереження прогресу. Дана діаграма дозволяє структурувати вимоги до функціоналу гри з точки зору кінцевого користувача.

Отже, основні вимоги до ігрового процесу:

– рівні, які мають створюватись процедурно з використанням шуму для забезпечення унікальності;

– гравець повинен мати можливість стріляти у будь-якому напрямку з колізіями та ефектами;

– вороги повинні мати особливу поведінку, патрулювання та адаптацію до дій гравця;

– вороги після знищення повинні залишати випадкові предмети, які можна підбирати;

– інтерфейс має відображати здоров’я, набої та активне завдання;

– користувач повинен мати змогу змінювати графіку, роздільну здатність та режим вікна;

– гра повинна мати головне меню з запуском, налаштуваннями та виходом.

Однією з ключових функцій гри є процедурна генерація карти, яка забезпечує унікальність ігрового середовища під час кожного запуску. Для цього планується використання комбінації різних алгоритмів. Шумові функції, які застосовуються для модифікації ландшафту, розподілу ресурсів, ворогів, а також формування рельєфу. Вони забезпечують натуральний, хаотично-структурований вигляд карти, водночас зберігаючи контрольовану складність.

Цей підхід дозволяє уникнути одноманітності рівнів, підвищити реіграбельність гри, а також зменшити витрати на ручну розробку кожної карти.

Бойова система є центральною механікою гри, що визначає взаємодію гравця з навколишнім середовищем та ворогами. Тому важливо сформувати вимоги до бойової системи:

– підтримка різної вогнепальної зброї, що дозволяє стріляти в усіх напрямках за допомогою миші;

– анімації стрільби, перезарядки та ефектів пострілу, які підвищують візуальну якість бойових дій;

– реалізація системи колізій для снарядів, що дозволяє враховувати перешкоди на рівні та блокування куль;

– можливість оновлення або покращення параметрів зброї.

Система має бути гнучкою для майбутнього розширення, додавання нового типу зброї або поведінки ворогів повинно відбуватись без значних змін у коді.

Інтерфейс користувача повинен забезпечити виведення всіх необхідних даних про поточний стан гри в реальному часі. Основні елементи HUD:

– смуга здоров’я гравця – вказує на залишок життя персонажа;

– поточне завдання – відображення цілей, які повинен виконати гравець;

– кількість набоїв в магазині та в запасі на даний момент.

Інтерфейс має бути адаптивним, лаконічним і витриманим у стилістиці гри, не перевантажуючи гравця зайвою інформацією.

З метою підвищення сумісності гри з різними системами та надання гравцеві можливості індивідуального налаштування, необхідно реалізувати меню налаштувань, яке включатиме вибір роздільної здатності екрану, перемикання між віконним і повноекранним режимом та зміна рівня якості графіки.

Меню повинно бути реалізоване у вигляді окремої сцени або вбудованого в гру діалогового вікна, з можливістю збереження налаштувань між сесіями гри.

Штучний інтелект (AI) у грі відіграватиме важливу роль у забезпеченні динамічного та цікавого геймплею. Основні вимоги до AI:

– адаптивність до гравця – вороги повинні змінювати свою поведінку в залежності від дій гравця (наприклад, укриття, атака, втеча);

– патрулювання;

– просторове розуміння карти – уникнення пасток, обходи перешкод, ефективна навігація (наприклад A* або NavMesh).

AI повинен бути реалізований як модульна система, щоб можна було легко додавати нові стани та логіку без переписування базового коду.

Після знищення ворогів, у грі має реалізовуватись система дропів – випадання предметів, які може підібрати гравець. Мають бути реалізовані різні типи предметів: боєприпаси, аптечки, тощо. Випадання предметів має базуватись на ймовірності та типі ворога. Гравець повинен мати можливість підібрати предмет.

Нефункціональні вимоги визначають характеристики програмного забезпечення, які не пов’язані безпосередньо з його функціональністю, проте суттєво впливають на зручність використання, продуктивність, сумісність та інші важливі аспекти.

Гра має бути реалізована таким чином, щоб була змога підтримувати декілька цільових платформ, у першу чергу на ПК як основна платформа для запуску та тестування гри. Рушій має забезпечувати високий рівень портованості, тому програмна архітектура проєкту повинна враховувати особливості кожної платформи вже на етапі розробки. Кросплатформність в майбутньому дозволить розширити аудиторію гри та зробити її доступною для більшої кількості користувачів.

Також потрібно забезпечити високу продуктивності гри на більшості сучасних пристроях, включаючи малопотужні:

– плавний геймплей при стабільній частоті кадрів на стандартних конфігураціях ПК;

– ефективне використання ресурсів: мінімізація споживання оперативної пам’яті та навантаження на графічний процесор;

– оптимізація графіки, шейдерів, колізій та обробки подій, особливо в контексті процедурної генерації та великої кількості об’єктів на сцені;

– масштабована якість графіки, яка дозволяє користувачу зменшити візуальні ефекти заради підвищення продуктивності.

Оптимізація має бути інтегрованою частиною розробки, що дозволить підтримувати гру на широкому спектрі пристроїв.

Структура програмного забезпечення повинна забезпечувати можливість подальшого розвитку гри без необхідності глобальних змін у кодовій базі. Сюди входить:

– додавання нових рівнів, локацій, типів ворогів, NPC, зброї та ігрових механік;

– розширення наявної системи бойової механіки, тощо;

– можливість легкої інтеграції нових модулів або плагінів Unity;

– застосування принципів модульного програмування, патернів проєктування, що забезпечують гнучкість коду та зменшення зв’язності між компонентами.

Таким чином, гра зможе розвиватися як проєкт з потенціалом до масштабування або комерціалізації.

Ігровий процес повинен бути інтуїтивно зрозумілим, а інтерфейс – простим та доступним навіть для новачків. Основні положення:

– інтуїтивне керування. Використання стандартних клавіш (WASD, миша) на ПК та сенсорних жестів для мобільних пристроїв;

– контекстна допомога, підказки, візуальні індикатори, що спрощують взаємодію з грою;

– мінімалістичний та зрозумілий UI з акцентом на читабельність і логічне групування елементів;

– відсутність перевантаження інформацією – поступове введення нових механік.

Комфорт користувача є одним із пріоритетних напрямів при розробці, адже від нього залежить залучення гравця та тривалість сесій. Нефункціональні вимоги доповнюють функціональні, визначаючи технічні та експлуатаційні характеристики гри. Вони сприяють створенню якісного програмного продукту, здатного працювати на різних пристроях, забезпечувати стабільний і комфортний ігровий процес, а також мати потенціал до подальшого розвитку.

Усі вимоги повинні бути враховані на етапі проєктування архітектури гри, що дозволить уникнути переробки на пізніших стадіях розробки.

2.2 Вибір засобів, методів і алгоритмів вирішення поставленого завдання

У процесі реалізації проєкту важливим етапом стало визначення оптимальних інструментів, методів та алгоритмів, які забезпечать ефективну розробку гри згідно з вимогами. Цей етап включає аналіз наявних технологій, порівняння їх переваг та недоліків, а також прийняття рішень щодо вибору найбільш доцільних засобів для досягнення поставленої мети. Зокрема, було здійснено оцінку кількох ігрових рушіїв, методів генерації ігрових рівнів, а також інструментів для створення графіки.

Для реалізації поставленого завдання, важливо обрати відповідне програмне середовище, яке дозволить ефективно працювати як над візуальною частиною, так і над логікою гри. На ринку існує багато ігрових рушіїв і фреймворків, зокрема Godot, Unreal Engine, GameMaker Studio, Defold та інші. Кожен із них має власні переваги та спеціалізації, тому вибір залежить від конкретних цілей і потреб розробника.

GameMaker Studio вважається одним із найпростіших у використанні для створення 2D-ігор, однак він обмежений у гнучкості, масштабованості та налаштуванні. Це чудовий варіант для невеликих проєктів або для початківців, але в рамках складнішої гри з генерацією світу, великою кількістю об’єктів та інтеграцією з іншими системами його можливостей може бути недостатньо.

Unreal Engine, хоча і підтримує 2D-ігри, значно краще оптимізований для 3D-проєктів [18]. Він потребує більших ресурсів, складніший в освоєнні та не завжди доцільний для проєктів середньої складності або інді-геймдеву. Крім того, розробка 2D-ігор на цьому рушії не настільки зручна, оскільки більшість функціоналу орієнтована на об’ємні сцени та тривимірну графіку.

Godot – це рушій з відкритим кодом, який останнім часом набуває популярності [9]. Він має зручне середовище розробки, легкий у навчанні та підтримує як 2D, так і 3D. Проте наразі він ще не досяг рівня зрілості та стабільності, який пропонують інші платформи, зокрема в плані підтримки сторонніх плагінів, документації, великої кількості уроків і активної спільноти.

Серед усіх перелічених варіантів Unity вирізняється як найбільш збалансований ігровий рушій для реалізації 2D-проєктів. Він підтримує візуальний редактор, компоненти фізики, роботу з анімацією, текстурами, а також інтеграцію з системами генерації контенту. Крім того, він має великий набір інструментів для тестування, налагодження й оптимізації гри, а також безліч готових рішень у магазині Asset Store.

Завдяки широкій базі знань, розвиненій спільноті, постійній підтримці оновлень і багатофункціональності, Unity став оптимальним вибором для реалізації проєкту. Він забезпечив достатню гнучкість для реалізації генерації світу, організації візуального стилю, налаштування взаємодії об’єктів та масштабування гри у майбутньому.

Мова програмування C# є основною мовою для розробки проєктів в Unity і забезпечує високу продуктивність, читабельність коду та об’єктно-орієнтований підхід, що ідеально підходить для створення масштабованих ігрових систем [6]. Вона має сучасний синтаксис, активно підтримується спільнотою та Microsoft, а також сумісна з великою кількістю бібліотек і плагінів. Завдяки інтеграції з середовищем Unity, C# дозволяє легко працювати з компонентною архітектурою, фізикою, анімацією, UI та іншими складовими ігрового рушія.

Крім того, C# має зручну систему обробки подій, делегатів і корутин, що дозволяє ефективно реалізовувати ігрову логіку, яка залежить від часу, наприклад, затримки між пострілами, анімації або чекпоінти. Завдяки використанню корутин у Unity, розробники можуть легко управляти асинхронними процесами без необхідності писати складний багатопоточний код.

Це значно спрощує реалізацію складних ігрових механік, зберігаючи при цьому стабільність та передбачуваність поведінки системи.

Ще однією перевагою є те, що C# дозволяє розробникам легко масштабувати кодову базу, розділяючи функціональність на окремі класи, скрипти та модулі. Це забезпечує високу підтримуваність проєкту, що є особливо важливим при довготривалій розробці або в командній роботі. Завдяки активній спільноті, великій кількості навчальних матеріалів та інтеграції з Visual Studio, C# виступає ідеальним інструментом для реалізації повноцінного ігрового проєкту в середовищі Unity.

Рушій має зручну систему компонування об’єктів за допомогою Tilemap, що особливо корисно при роботі з процедурною генерацією рівнів на основі тайлів. Також має інтуїтивно зрозумілий інтерфейс для розробки, який показано на рисунку 2.2.

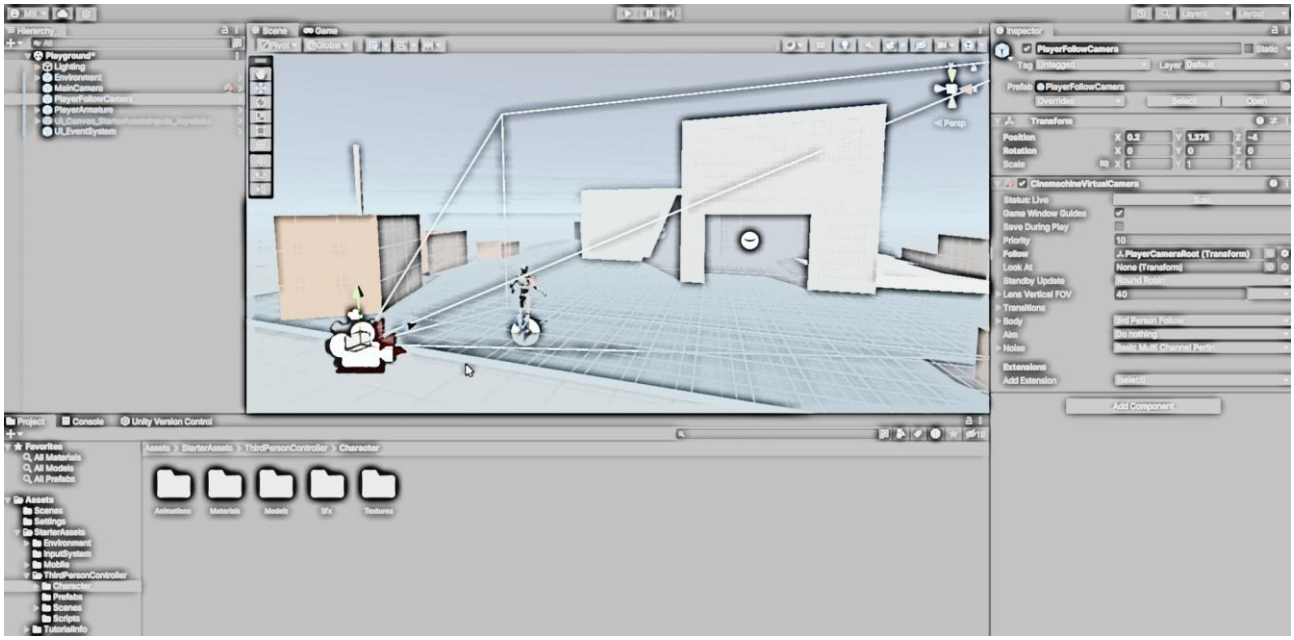

Рисунок 2.2 – Інтерфейс Unity

Одним із ключових елементів проєкту є реалізація генерації рівнів. Для цього було розглянуто та проаналізовано декілька варіантів алгоритмів, випадкове розміщення об’єктів, сіткову генерацію, заздалегідь підготовлені шаблони та генерацію на основі шумових функцій. Кожен з цих підходів має свої

особливості. Наприклад, випадкове розміщення забезпечує непередбачуваність, але може призводити до хаотичної, нелогічної структури. Шаблонна генерація дозволяє краще контролювати форму рівня, але обмежує варіативність.

У ході аналізу було визначено, що найбільш оптимальним варіантом для досягнення балансу між структурованістю й випадковістю є використання шуму Перліна. Цей підхід дозволяє створювати плавні, природні переходи між областями, що добре підходить для генерації середовища на відкритих просторах – наприклад, для визначення густоти лісу, розміщення галявин або розподілу рослинності. Перлін-шум генерує значення в межах від 0 до 1, які змінюються поступово, що й забезпечує природність у результаті.

Основною перевагою шуму Перліна є його «контрольований хаос» – він зберігає випадковість, але дозволяє уникнути різких змін або непередбачуваних зон. Це критично важливо для створення ігрових локацій, де важливо дотримуватися певного візуального порядку або забезпечити логіку розміщення об’єктів. Завдяки цьому, гравець сприймає рівень як цілісний світ, а не як набір розкиданих елементів.

Таким чином, вибір генерації середовища на основі шуму Перліна був зумовлений потребою у створенні плавних, логічних локацій зі збереженням випадковості. Це дозволило забезпечити унікальність кожної карти при кожному запуску гри, не втрачаючи при цьому цілісності ігрового простору.

Одним із важливих аспектів при розробці гри є забезпечення інтелектуальної поведінки неігрових персонажів, зокрема їх здатність орієнтуватися у просторі рівня, уникати перешкод та досягати цілей, таких як гравець. Для вирішення цього завдання було проаналізовано кілька підходів до реалізації навігації та прийнято рішення використати навігаційну сітку NavMesh, яка є стандартним і потужним інструментом у середовищі Unity [17].

Попри те, що NavMesh спочатку створювався для 3D-простору, його успішно адаптовано для 2D-проєктів шляхом проєкції сцени на площину (наприклад, X-Y або X-Z), використання ортографічної камери та коректного налаштування об’єктів (рис. 2.3). Це дозволяє застосовувати всі основні переваги:

автоматичне планування маршруту, уникнення перешкод, динамічне оновлення сітки та просте масштабування.

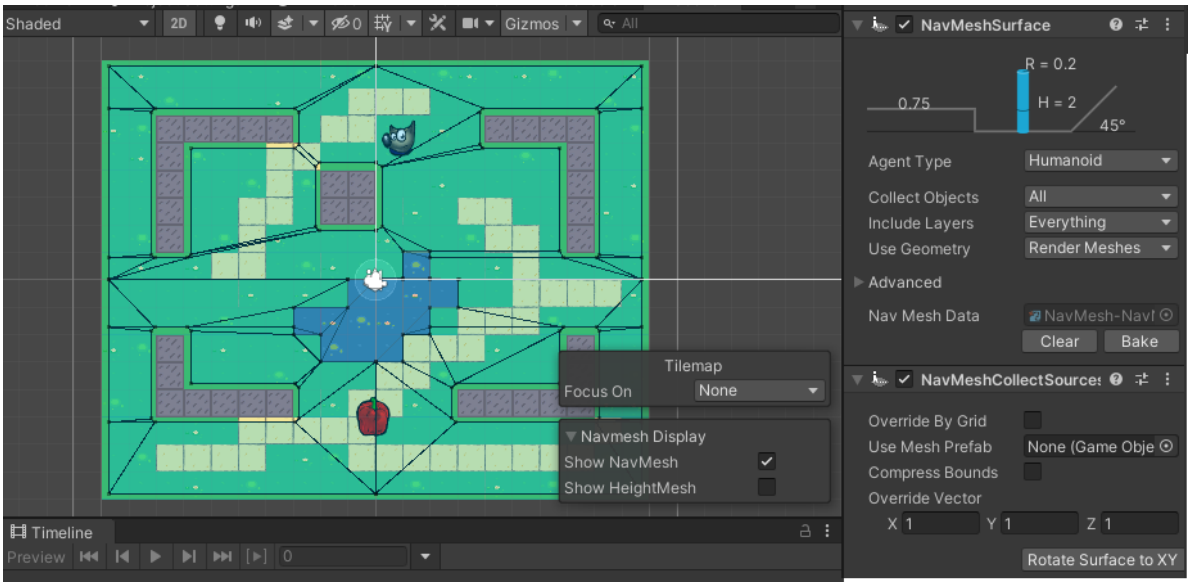

Рисунок 2.3 – Сітка NavMesh

Було також розглянуто альтернативні підходи:

– A (A-Star)* – один із найпоширеніших алгоритмів пошуку шляху [2]. Його реалізація вимагає ручного побудування графа або сітки рівня, створення вузлів та ребер, а також логіки обходу. Цей метод дає повний контроль над маршрутом, але вимагає значних зусиль на етапі реалізації та підтримки, особливо при змінній геометрії рівня;

– системи на основі векторного переміщення (Steering Behaviors) – застосовуються для імітації природної поведінки персонажів (рух до цілі, уникнення зіткнень, слідування за групою тощо) [4]. Вони добре підходять для простих сценаріїв, але не забезпечують побудови повноцінного шляху через складні лабіринти або приміщення;

– графові підходи з ручним плануванням кімнат – передбачають побудову логічної карти рівня. Підходять для рівнів із фіксованою структурою, але обмежені в гнучкості, особливо при використанні процедурної генерації.

У порівнянні з цими варіантами, NavMesh в Unity надає баланс між автоматизацією та функціональністю. Він дозволяє реалізувати складну логіку руху з мінімальними витратами часу, а також легко адаптується під зміни у сцені. За потреби можливе використання сторонніх рішень, що адаптують NavMesh для 2D – наприклад, NavMesh Plus, який підтримує Tilemap і дозволяє генерувати сітку прямо на 2D рівні.

Таким чином, вибір на користь NavMesh як засобу для реалізації системи навігації у грі був обґрунтований його високою ефективністю, простотою інтеграції з Unity, підтримкою динамічних змін і оптимальною продуктивністю для ігор у 2D-середовищі.

Важливою складовою ефективного процесу розробки програмного забезпечення є використання системи контролю версій. У межах реалізації цього проєкту було прийнято рішення використовувати Git як основну систему керування версіями, а також хмарний сервіс GitHub як платформу для зберігання, обміну та співпраці над кодом [8].

GitHub забезпечує низку переваг для організації розробки:

– збереження історії змін дозволяє відслідковувати всі внесені правки до коду, повернутись до попередньої версії у разі виникнення помилок або необхідності відкату;

– гілкування дає змогу працювати над окремими частинами проєкту (наприклад, генерація рівнів, бойова система, інвентар) незалежно, не порушуючи основну гілку розробки;

– Pull Requests дають змогу перевіряти та переглядати зміни перед об’єднанням у головну гілку, що особливо корисно при роботі в команді або під час перевірки функціональності нових модулів;

– Issue Tracker дозволяє створювати задачі, відслідковувати помилки, пропозиції щодо покращення і вести організовану документацію до процесу розробки;

– інтеграція з CI/CD сервісами може бути використана у майбутньому для автоматизації збірки гри, тестування або деплою.

У межах проєкту буде створений окремий GitHub-репозиторій, у якому будуть зберігатися всі основні файли проєкту, включно зі сценами Unity, скриптами на C#, ресурсами, а також документація. Кожна ключова зміна супроводжуватиметься описовим комітом, що дозволить підтримувати структурованість та зрозумілість коду протягом усього періоду розробки.

Завдяки використанню GitHub забезпечується не лише збереження стабільної версії проєкту, а й підвищується зручність співпраці, тестування та розширення гри в майбутньому.

У процесі створення гри з піксельною графікою постає питання вибору джерела спрайтів та інструментів для їх створення. В магазині Asset Store для Unity доступна велика кількість готових наборів піксельних текстур, персонажів, анімацій і наборів. Вони можуть значно пришвидшити розробку на ранньому етапі та допомогти сформувати базовий візуальний стиль проєкту. Проте більшість з них мають загальний характер, і, щоб досягти бажаного, все одно виникає потреба у створенні власних елементів.

Саме тому, попри наявність великої бібліотеки, більшість графічного контенту доведеться створювати вручну. Це дозволить точно відповідати стилістиці гри, її естетиці та геймплейним особливостям. Для цього потрібно мати відповідне програмне забезпечення, зручне для створення саме піксельної графіки. Найбільш поширеними інструментами у цій сфері є Adobe Photoshop або Aseprite.

Adobe Photoshop – це потужний графічний редактор із широким функціоналом для роботи з зображеннями будь-якого типу [1]. Він підтримує анімацію, шари, маски та інші просунуті функції. Проте для створення саме піксель-арту він вважається занадто складним і перевантаженим зайвими функціями. До того ж, його інтерфейс не орієнтований на роботу з піксельною сіткою.

На відміну від нього, Aseprite є спеціалізованим редактором піксельної графіки [3]. Він має простий та інтуїтивно зрозумілий інтерфейс зображений на рисунку 2.4, також інструменти, що оптимізовані для піксельного малювання,

зручну систему анімації кадрів і підтримку наборів текстур. Крім того, він значно легший та швидше працює. Саме тому Aseprite був обраний основним інструментом для створення індивідуальних спрайтів, анімацій та інших елементів піксельної графіки, необхідних для гри.

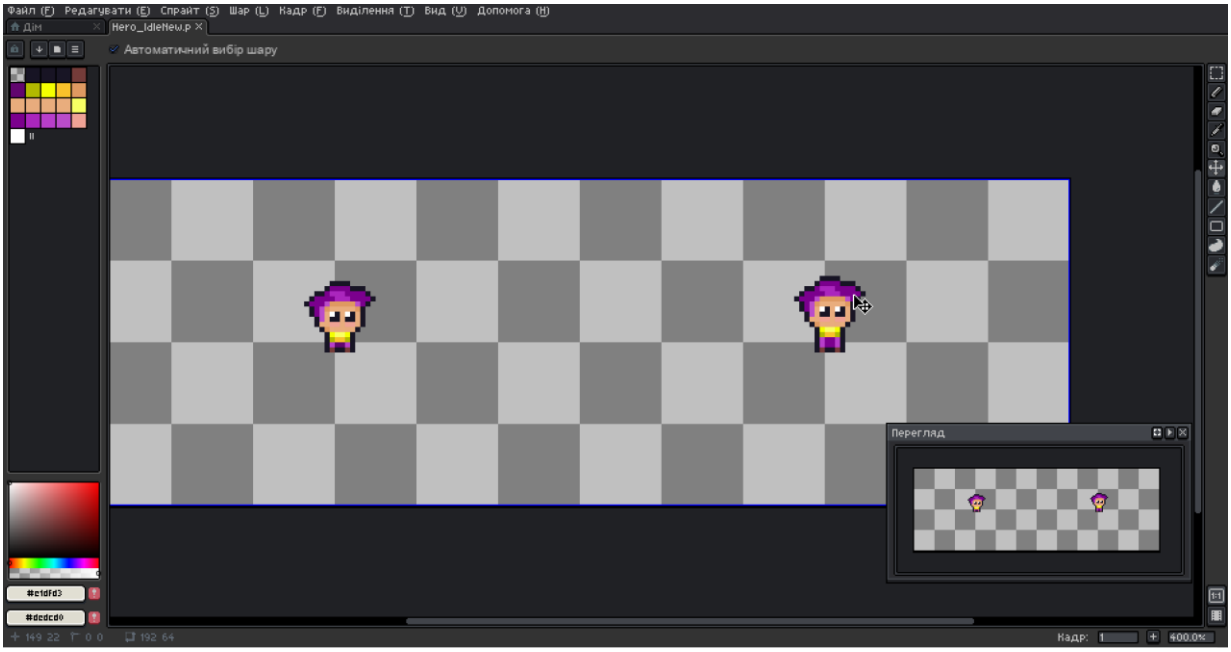

Рисунок 2.4 – Інтерфейс Aseprite

Для організації робочого процесу, планування задач і контролю за виконанням етапів розробки також було використано інструмент Trello [15]. Цей онлайн-сервіс надає зручну візуальну форму у вигляді дошки з картками, які можна розподіляти за категоріями. Такий підхід дозволяє чітко бачити поточний стан проєкту, розставляти пріоритети, планувати дедлайни та розподіляти час на виконання конкретних завдань.

У межах проєкту в Trello будуть створені окремі списки для технічних завдань (наприклад, реалізація процедурної генерації, створення інтерфейсу, бойової механіки), а також для візуальних і дизайнерських задач (робота зі спрайтами, UI/UX, тестування рівнів). Кожна картка містить опис, вимоги із пунктами та позначки для швидкої орієнтації за категоріями (рис. 2.5).

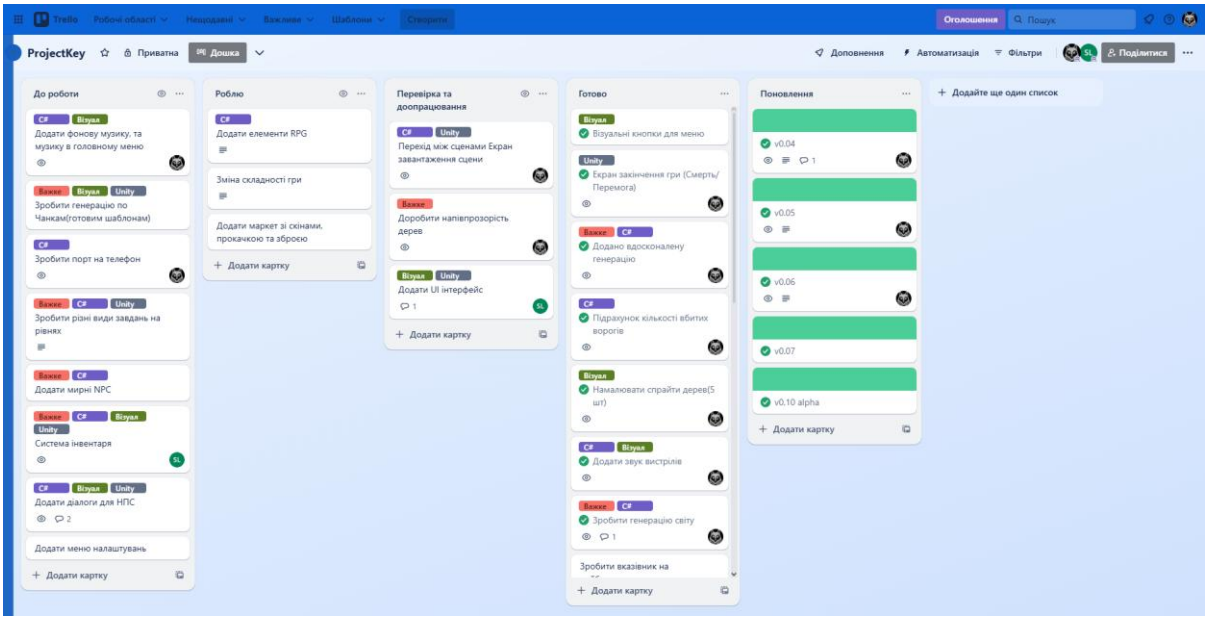

Рисунок 2.5 – Дошка з завданнями в Trello

Загалом, Trello виступить як інструмент для самоорганізації та управління проєктом, дозволить систематизувати роботу і не втратити важливі деталі під час активної фази розробки. Його простота та доступність зробить його зручним доповненням до технічних інструментів, таких як Unity, Git та GitHub.

РОЗДІЛ 3
РОЗРОБКА КОМП'ЮТЕРНОЇ ГРИ

3.1 Практична реалізація об'єкта проектування

Для наочного представлення процесу вибору технічних рішень розробником, створено UML-діаграму варіантів використання зображену на рисунку 3.1.

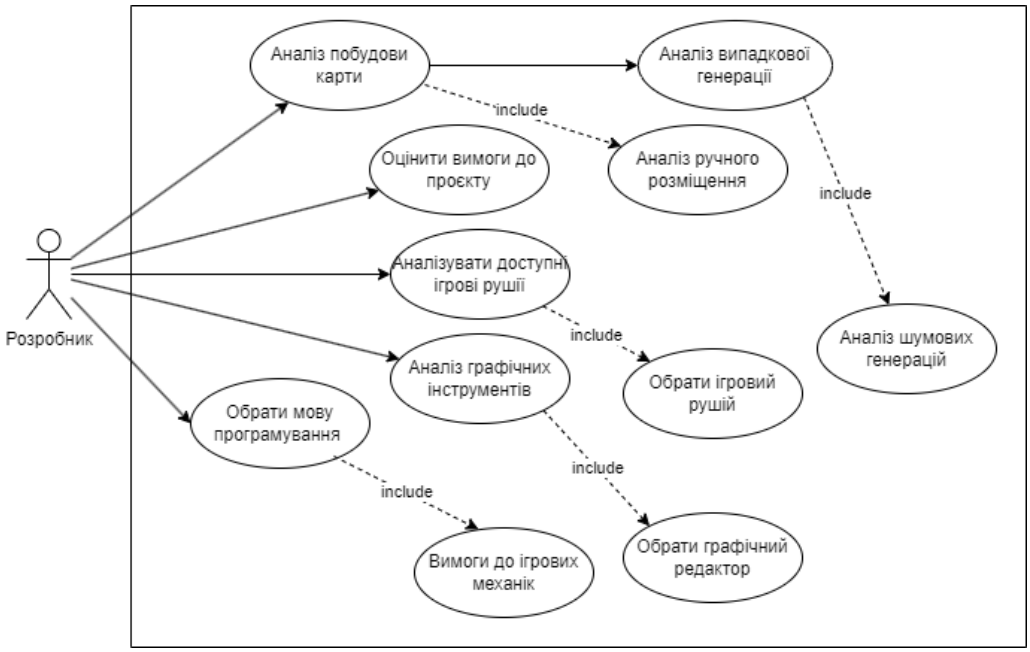

Рисунок 3.1 – UML діаграма

Діаграма демонструє, які дії виконує розробник на даному етапі, з якими задачами він взаємодіє, і як відбувається вибір інструментів та алгоритмів на основі аналізу вимог до проєкту. Дана діаграма допомагає структурувати процес прийняття рішень і візуалізує послідовність дій, що передували остаточному вибору програмного забезпечення, методів генерації контенту та інструментів розробки.

У межах розробки гри важливим етапом стала реалізація системи керування гравцем. Замість використання застарілих методів обробки натискань клавіш через клас Input, було прийнято рішення застосувати сучасну систему введення Unity – Input System Package із використанням PlayerInputActions. Це

надало більшу гнучкість, розширюваність та покращену інтеграцію з різними платформами і пристроями введення.

Система PlayerInputActions дозволяє створити окремий файл вводу, в якому описуються дії гравця, а також їх прив’язка до різних типів пристроїв (клавіатура, геймпад тощо). Для реалізації руху було створено дію, яка очікує значення типу Vector2 – це дозволяє одразу отримувати напрямок руху по осях X та Y у вигляді єдиного вектору.

Процес реалізації руху складався з кількох ключових етапів:

1) налаштування Input Actions Asset. У редакторі Unity було створено новий Input Actions Asset. У ньому додано карту дій під назвою Player, в яку включено дію Move, що обробляє WASD, стрілки або аналогові стіки. Тип дії було задано як Value з типом Vector2 (рис. 3.2);

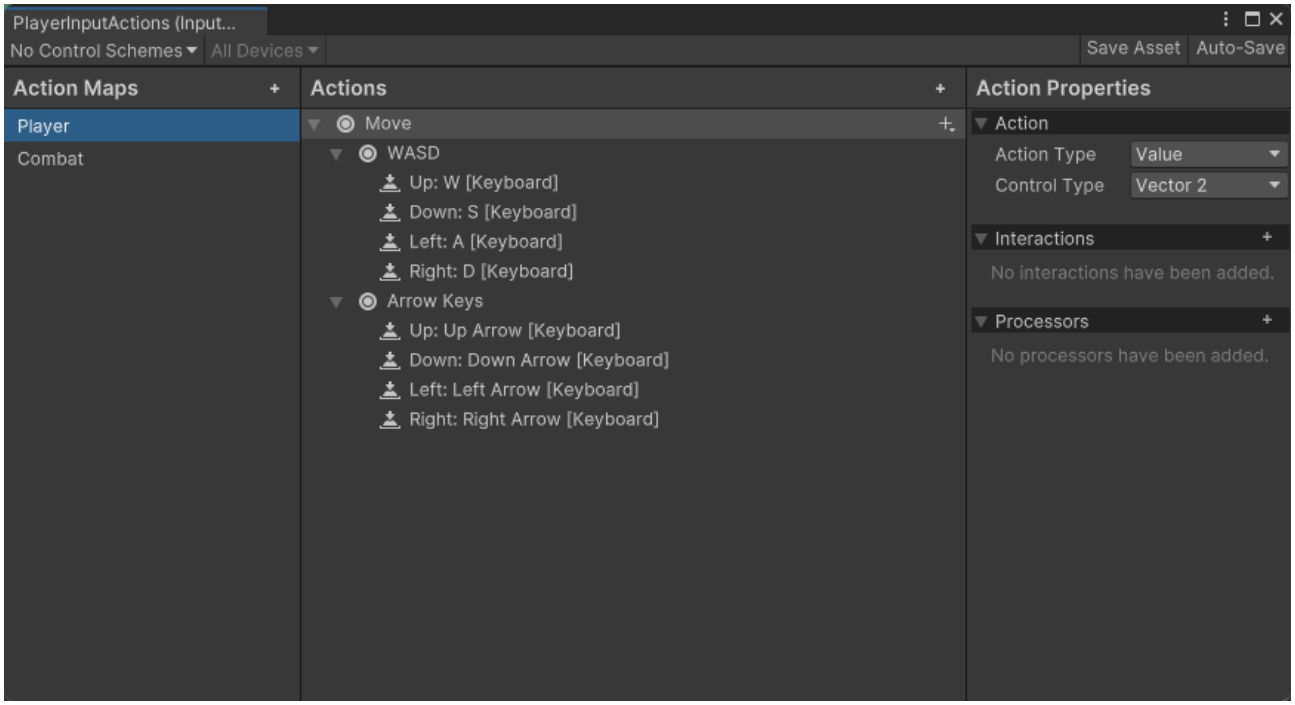

Рисунок 3.2 – Меню Input System

2) генерація класу PlayerInputActions у Unity відбувається автоматично і містить усі визначені дії. Цей клас дозволяє зручно підписуватись на події або зчитувати значення напряму;

3) створення компоненту PlayerMovement, який підключає PlayerInputActions та реалізує логіку переміщення. Вектор руху зчитується з Input System у кожному кадрі та застосовується до компонента Rigidbody2D або Transform. Таке рішення дозволяє у майбутньому з легкістю додавати інші дії (наприклад, Jump, Attack, Interact) без написання великої кількості умов. Також зручно підтримуються різні платформи: керування на геймпаді, мобільному пристрої чи ПК вмикається автоматично в залежності від вводу;

4) реалізація вектора руху, щоб автоматично змінювати анімації персонажа в залежності від напряму руху. Він буде переданий до Animator для подальшої реалізації анімацій;

5) впровадження подієвого підходу, наприклад, виклик певної функції лише під час натиснення чи відпускання клавіші. Це зручно для інших дій, таких як взаємодія, атака, або відкриття інвентарю. Система Input Actions підтримує подієвий підхід.

Таким чином, використання PlayerInputActions забезпечує сучасну, чисту та гнучку архітектуру керування в грі. На відміну від застарілих методів, нова система дозволяє централізовано керувати усім введенням, легко обслуговувати зміни та масштабувати гру під майбутні вимоги.

Однією з основних ігрових механік, реалізованих у рамках проєкту, є механіка стрільби, яка забезпечує взаємодію гравця з ворожими об’єктами та навколишнім середовищем. Для її реалізації було створено спеціальний скрипт, який відповідає за поведінку зброї як у руках гравця, так і у ворожих персонажів.

Механізм стрільби включає низку складових: створення об’єкта кулі, обробку напрямку прицілювання, обмеження за кількістю патронів, відлік часу між пострілами, а також систему перезарядки та візуального супроводу.

У випадку керування гравцем, напрям стрільби визначається на основі позиції курсора миші. Для цього обчислюється вектор від об’єкта зброї до позиції курсора на екрані, після чого за допомогою функції Mathf.Atan2 визначається кут повороту, що застосовується до об’єкта. У разі ворогів, орієнтація зброї базується на положенні гравця на сцені.

Стрільба можлива лише за дотримання кількох умов:

– користувач натиснув кнопку стрільби (ліва кнопка миші);

– персонаж не знаходиться у стані перезарядки;

– у поточному магазині є боєприпаси;

– від моменту останнього пострілу пройшов достатній проміжок часу.

При виконанні вищезазначених умов відбувається створення об’єкта кулі за допомогою методу Instantiate(), який створює екземпляр кулі у вказаній точці з відповідною орієнтацією, зображеній на рисунку 3.3. Одночасно активується звуковий ефект пострілу, оновлюються параметри анімації та зменшується кількість наявних патронів.

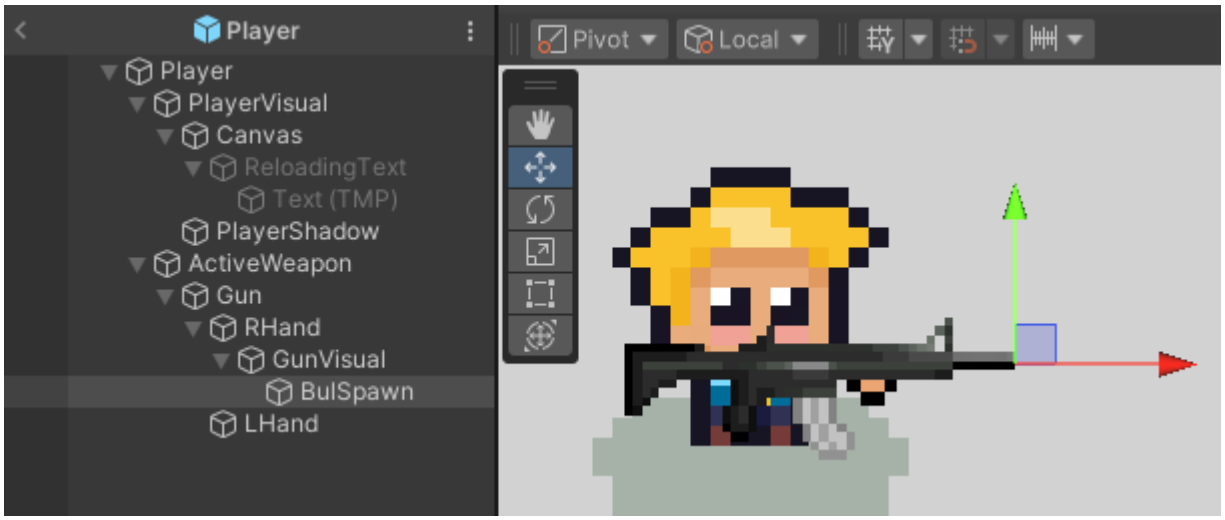

Рисунок 3.3 – Механізм стрільби

Проміжок часу між пострілами контролюється змінною, яка скидається до початкового значення після кожного пострілу та поступово зменшується у циклі.

У разі закінчення патронів або при натисканні клавіші R ініціюється процес перезарядки, який реалізовано у вигляді корутини наведеній на лістингу 3.1. Під час перезарядки активується графічний індикатор та програється відповідний звуковий ефект, після чого виконується дозаправка магазину з загального запасу патронів. При цьому забезпечується перевірка на наявність боєприпасів та запобігання перевищенню максимального ліміту.

Лістинг 3.1 – Корутина перезарядки зброї

```
private IEnumerator Reload()
{
    reloadingText.gameObject.SetActive(true);
    isReloading = true;
    if (audioReloading)
        audioGun.PlayOneShot(audioReloading);
    yield return new WaitForSeconds(reloadTime);
    int reason = maxCurrentAmmo - currentAmmo;
    if (allAmmo >= reason)
    {
        allAmmo = allAmmo - reason;
        currentAmmo = maxCurrentAmmo;
    }
    else
    {
        currentAmmo = currentAmmo + allAmmo;
        allAmmo = 0;
    }

    isReloading = false;
    reloadingText.gameObject.SetActive(false);
}
```

кінець лістингу 3.1

Візуалізація стану боєприпасів реалізована за допомогою текстового елемента інтерфейсу, який у режимі реального часу оновлює інформацію про кількість патронів у магазині та загальний запас. Це підвищує зручність користування та покращує взаємодію з гравцем.

Крім того, передбачено логіку підбору боєприпасів через обробку подій зіткнення з об’єктами типу. Після підбору патрони додаються до загального запасу, з урахуванням обмеження на максимальну кількість.

Узагальнюючи, реалізація стрільби базується на гнучкій, модульній архітектурі, яка дозволяє масштабувати механіку на інші типи зброї, легко адаптувати її під нових ворогів, а також забезпечує високий рівень інтерактивності та керованості з боку гравця. Важливо, що вся функціональність узгоджена із загальною структурою проєкту та враховує потреби користувача щодо візуального та звукового зворотного зв’язку.

Для реалізації плавного та зручного керування камерою в проєкті було обрано використання інструменту Cinemachine, який є стандартним пакетом у середовищі розробки Unity. Cinemachine значно полегшує створення динамічної поведінки камери без необхідності написання великої кількості власного коду.

У 2D-проєкті Cinemachine дозволяє реалізувати слідкування камери за гравцем із можливістю згладження руху, що забезпечує більш природну динаміку кадру. Крім того, інструмент надає можливість встановлення меж області перегляду, щоб камера не виходила за межі рівня, а також підтримує зміну фокусування між різними об’єктами, що корисно для ігрових сцен із перемиканням уваги.

Використання Cinemachine Virtual Camera дозволяє легко призначити об’єкт стеження (наприклад, гравця), налаштувати швидкість реакції камери на переміщення та додати ефекти, як-от легке коливання під час пострілу або удару, що підвищує рівень занурення у гру.

У межах реалізації головного ігрового персонажа було створено функціональність, що забезпечує переміщення, обробку отримання шкоди, взаємодію з об’єктами навколишнього середовища, оновлення інтерфейсу здоров’я, а також звуковий супровід дій гравця.

Для керування життям гравця реалізована система, що включає максимальний та поточний рівень здоров’я. У разі отримання пошкоджень здоров’я зменшується на відповідну кількість одиниць. Візуальне відображення змін реалізовано за допомогою елементу інтерфейсу – шкали здоров’я, яка оновлюється у реальному часі.

Щоб уникнути постійного отримання шкоди, реалізовано затримку між можливостями завдання наступного удару, яка забезпечується за допомогою таймера. Також додано візуальний ефект при пошкодженні у вигляді частинок, що імітують кров, що підвищує візуальну виразність сцени.

У разі, якщо здоров’я зменшується до нуля, персонаж вважається мертвим. У цьому випадку вимикається керування та ігровий процес призупиняється через виклик відповідного функціоналу, який відображає екран завершення гри. Для

інших частин гри реалізовано подію, що сповіщає про смерть гравця, дозволяючи іншим об’єктам реагувати на неї.

На лістингу 3.2 показаний код який відповідає за нанесення шкоди гравцю, а також за виклик інших вище описаних функцій.

Лістинг 3.2 – Функція TakeDamage

```
public void TakeDamage(int damage)
{
    if (_canTakeDamage && _isAlive)
    {
        _canTakeDamage = false;
        _curentHealth = Mathf.Max(0, _curentHealth -= damage);
        OnTakeHits?.Invoke(this, EventArgs.Empty);

        Instantiate(bloodPrefab,                    transform.position,
Quaternion.identity);
        StartCoroutine(DamageRecoveryRoutine());
    }

    DetectDeath();
}
```

кінець лістингу 3.2

Ігровий персонаж здатен взаємодіяти з об’єктами довкілля, наприклад аптечками. При зіткненні з ними відбувається відновлення частини здоров’я, але не більше ніж максимально допустиме значення. Після взаємодії об’єкт аптечки видаляється зі сцени, що імітує його використання.

Було реалізовано події, які активуються у момент отримання пошкоджень або смерті гравця. Це забезпечує можливість масштабування логіки гри без необхідності змінювати основний скрипт. Також доступна функціональність, яка дозволяє визначити координати персонажа на екрані для синхронізації з інтерфейсними елементами.

Таким чином, реалізація функціоналу керованого персонажа забезпечує не лише базову логіку руху та виживання, але й містить гнучку архітектуру, що дозволяє зручно розширювати її в подальшому. Інтеграція з іншими системами

гри (інтерфейсом, аудіо, фізикою) виконується через події та інкапсульовані механізми, що відповідає принципам якісного програмного дизайну.

У межах реалізації ігрових ворогів було використано систему штучного інтелекту на базі машини станів та NavMesh Agent, яка забезпечує коректну навігацію по ігровому середовищу. Такий підхід дозволяє створити адаптивну та реалістичну поведінку ворогів, які можуть патрулювати територію, переслідувати гравця, атакувати його або перебувати в стані очікування.

Вороги у грі переходять між кількома базовими станами:

– Idle (стан очікування) – коли ворог не виконує жодних дій;

– Roaming (патрулювання) – рух в межах заданого радіусу від початкової позиції, з випадковим вибором напрямку;

– Chasing (переслідування) – активне переслідування гравця, якщо той потрапляє в зону виявлення;

– Attacking (атака) – потенційна атака при наближенні до гравця на відповідну відстань;

– Death (смерть) – деактивація логіки після знищення ворога.

Для кожного з цих станів передбачено окрему поведінкову логіку, яка активується через метод обробки станів. Переходи між станами виконуються динамічно, залежно від відстані до гравця та поточних умов.

Для реалізації руху ворогів було застосовано NavMeshAgent, що дозволяє здійснювати навігацію в межах навігаційної сітки. Використання NavMesh забезпечує:

– уникнення перешкод під час руху;

– плавну зміну напрямку;

– коректний розрахунок маршруту до цілі.

Сама сітка попередньо генерується в редакторі Unity на базі геометрії сцени, і враховує всі прохідні та непрохідні зони, а також оновлюється після генерації карти в реальному часі та виділяє усі потрібні об’єкти (рис. 3.4).

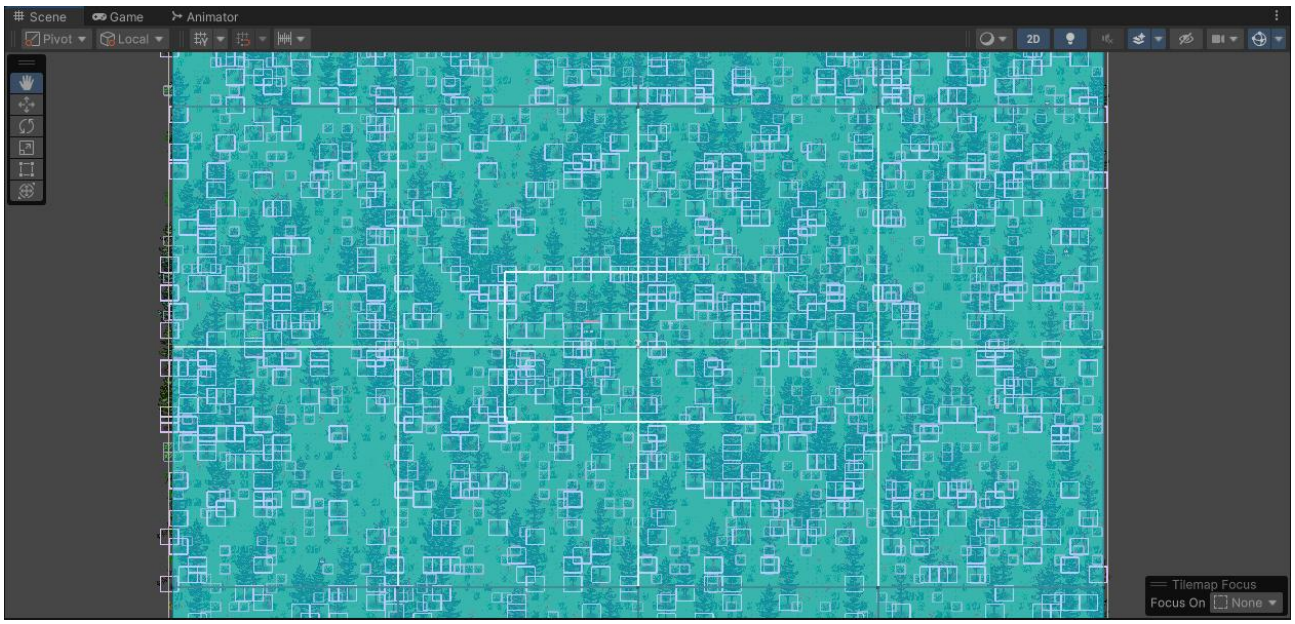

Рисунок 3.4 – Сітка NavMesh після генерації карти

Навігаційний агент ворога використовує методи, щоб переміщатися до гравця або випадкової точки при патрулюванні. В залежності від режиму ворога змінюється його швидкість, при патрулюванні використовується стандартна швидкість, а при переслідуванні гравця – пришвидшена. Це дозволяє ворогам виглядати більш динамічними.

Щоб вороги завжди були зорієнтовані в бік руху або гравця, реалізовано зміну напряму погляду через переворот спрайта. Це здійснюється шляхом обертання об’єкта по осі Y залежно від положення цілі відносно ворога.

Отже, реалізація ворогів базується на простій, але гнучкій системі штучного інтелекту з використанням NavMesh, що дозволяє забезпечити природну поведінку супротивників та їхню взаємодію з гравцем у різних сценаріях гри.

У грі, реалізована система процедурної генерації ігрового світу – карти, яка складається з землі, доріг, будівель, рослинності та декоративних елементів, зокрема трави. Цей підхід дозволяє автоматизовано створювати великі, світи без необхідності вручну компонувати кожен елемент. Генерація карти здійснюється під час запуску сцени за допомогою окремого скрипта, що використовує

різноманітні методи математичного моделювання та логіки розташування об’єктів у просторі.

На першому етапі ігрове поле заповнюється основним шаром тайлів, які імітують землю або траву. Зазвичай це квадратна ділянка з координатами, що охоплюють певний діапазон (наприклад, від -50 до 50 по обох осях). На кожну клітинку карти випадковим чином вибирається один із доступних тайлів трави, щоб надати поверхні більш природного та нерівномірного вигляду. Це дозволяє уникнути ефекту повторення одного й того ж візерунка, що є важливим для створення візуально привабливого ігрового середовища.

Після створення шару землі додається дорога. Вона може бути або вертикальною, або горизонтальною – напрямок вибирається випадково. Дорога має певну ширину, яка встановлюється як параметр, і проходить через карту повністю по відповідній осі. На ділянках дороги замінюється базовий тайл землі на спеціальні тайли дороги, які теж можуть бути обрані випадково з наявного набору. Такий підхід дозволяє створити різні візуальні варіації дороги при кожному запуску гри.

На основі псевдовипадкових координат система намагається розмістити певну кількість будівель на карті. Однак перш ніж створити об’єкт, скрипт перевіряє, чи ця ділянка не зайнята дорогою, іншою будівлею або чи не перебуває надто близько до цих об’єктів. Для цього проводиться перевірка в радіусі заданої відстані. Лише у разі проходження всіх перевірок, будівля з обраного заздалегідь списку префабів створюється в заданому місці. Це дозволяє уникнути хаотичного і недопустимого розміщення об’єктів.

Найцікавішою частиною генерації середовища є розміщення рослинності, зокрема дерев, яке базується на алгоритмі шуму Перліна. На відміну від звичайної генерації випадкових значень, шум Перліна створює плавні переходи між значеннями, що ідеально підходить для природних ландшафтів.

Для кожної клітинки карти обчислюється координата шуму за допомогою функції Mathf.PerlinNoise, яка приймає два значення (x та y координати) і

повертає число від 0 до 1. Отримане значення порівнюється з визначеними порогами. Наприклад:

– якщо значення шуму > 0.9 – вважається, що це лісиста місцевість, і тут велика ймовірність появи дерев;

– якщо значення шуму < 0.7 – це галявина або відкрита місцевість, дерева там зустрічаються рідко;

– якщо значення шуму знаходиться між порогами – це перехідна зона з помірною кількістю дерев.

Таким чином, карта природним чином розділяється на густі ліси, відкриті поля та змішані зони. Це надає грі візуальної глибини та робить кожну згенеровану карту унікальною.

Шум Перліна є функцією, що дозволяє згенерувати випадкові значення у двовимірному або тривимірному просторі. На відміну від звичайного генератора випадкових чисел, який створює незалежні та хаотичні значення, шум Перліна створює перехідні області, де значення змінюються поступово. Це дуже схоже на те, як виглядають природні структури – хмари, гори, текстури землі, тощо.

Основна ідея полягає в тому, що у певній сітці простору (наприклад, на кожній цілій координаті) генеруються вектори-градієнти. Потім значення в точці між цими координатами обчислюється як інтерпольована сума добутків цих градієнтів на вектори напрямку до точки. Таким чином, значення в одній точці залежить від сусідніх, що й забезпечує плавність.

У Unity використовується функція Mathf.PerlinNoise(float x, float y), яка реалізує цей алгоритм для 2D простору та повертає значення в діапазоні [0; 1]. Змінюючи вхідні значення або масштаб координат через коефіцієнт noiseScale, можна регулювати рівень деталізації й різноманітності отриманих візерунків.

Останнім етапом є генерація дрібних декоративних елементів, зокрема травинок. Випадковим чином вибираються координати на карті, і, якщо обрана клітинка не є частиною дороги, в неї вставляється одна з доступних моделей трави. Це дозволяє зробити карту більш живою і деталізованою, без суттєвого впливу на геймплей.

Переваги процедурної генерації:

– кожна карта є неповторною, що підвищує реіграбельність гри;

– легко розширювати карту без ручного редагування;

– можна додавати нові типи об’єктів або зон, змінюючи лише логіку генерації;

– зменшується обсяг роботи дизайнера при створенні рівнів.

Цей підхід забезпечує високий рівень автоматизації при створенні ігрового світу, знижує потребу в ручному розміщенні об’єктів, а також відкриває можливості для динамічної генерації середовища під час гри. У майбутньому систему можна адаптувати до створення підземель, міст або навіть погодних умов, використовуючи ті самі принципи генерації з перлін-шумом і додатковими логічними обмеженнями.

Розробка головного меню стала важливою складовою проєкту, оскільки саме з нього починається взаємодія з грою. Меню реалізоване як окрема сцена в Unity та містить основні елементи навігації: кнопки «Грати», «Магазин», «Опції», «Вихід», а також декоративні елементи у стилістиці гри (рис. 3.5).

Рисунок 3.5 – Головне меню

Кожна кнопка має анімацію при натисканні, що покращує користувацький досвід і робить меню більш живим. Для створення логіки перемикання сцен використовувався C# скрипт, який обробляє події натискання на кнопки та завантажує ігрову сцену чи виконує вихід з програми.

Меню магазину було поділено на дві основні вкладки – «Покращення» та «Скіни». У вкладці покращень гравець може підвищити основні характеристики персонажа: шкоду зброї, кількість здоров’я та місткість магазину. Для кожного параметра передбачено кнопки оновлення та індикатор поточного значення.

У вкладці скінів відображається набір доступних зовнішніх виглядів персонажа. Кожен скін відображається у вигляді карточки та можна придбати за внутрішню валюту гри. Після покупки гравець може активувати бажаний скін, і зміни негайно застосовуються до моделі персонажа в грі, також зберігаються усі покращення та покупки гравця, тому скіни можна застосувати коли завгодно.

Меню налаштувань, доступне з головного меню, дозволяє гравцеві змінювати графічні параметри, рівень важкості та переглядати статистику гри. Всі налаштування зберігаються між сесіями за допомогою системи PlayerPrefs, що дозволяє користувачу зберегти свої вподобання.

Загалом, головне меню було реалізоване як інтуїтивно зрозумілий, функціональний і візуально привабливий компонент, що виконує як технічну, так і естетичну функцію у грі.

3.2 Тестування та налагодження інформаційно-комп’ютерної системи

Після завершення основної розробки гри було проведено тестування з метою перевірки стабільності, працездатності та відповідності функціоналу поставленим вимогам. Тестування здійснювалося на різних етапах розробки – як модульно, так і інтегровано, що дозволило виявити та усунути критичні помилки ще на ранніх етапах.

Окрему увагу було приділено генерації рівнів за допомогою шуму Перліна. Було перевірено, чи коректно генеруються елементи залежно від від обраного біому, а також чи не виникає колізій між об’єктами. Для цього було реалізовано візуальний режим перегляду згенерованої карти із відображенням координат і типів блоків.

Ігрова механіка також пройшла функціональне тестування – перевірено роботу зброї, бойової системи, а також взаємодії з ворогами. Було протестовано системи збереження та завантаження прогресу, переміщення між сценами, відкриття вікон, відображення здоров’я, боєприпасів та досвіду гравця.

Значну увагу було приділено тестуванню UI – перевірено коректність відображення кнопок, меню, магазинів, екрану паузи, а також довідки. Усі UI-елементи адаптовано до різних роздільностей екранів, що забезпечує однакову зручність користування на різних пристроях.

Також проводилось тестування продуктивності – перевірялась стабільність кадрів при великій кількості об’єктів на сцені, особливо в складних генераціях. Була оптимізована кількість викликів фізичних та графічних обчислень, анімацій і використання текстур.

Для зовнішнього тестування було залучено невелику групу користувачів, які надали відгуки щодо ігрового процесу, зручності керування та можливих багів. Усі користувачі мали різний досвід у відеоіграх – від новачків до гравців зі значним стажем. Такий підбір учасників дозволив отримати максимально об’єктивну оцінку з різних точок зору. Тестувальники грали в попередньо зібрану версію гри, проходили кілька рівнів, взаємодіяли з інтерфейсом, магазинами, ворогами та перевіряли механіку генерації локацій.

Кожному учаснику було запропоновано пройти гру кілька разів у різних умовах і заповнити анкету з враженнями, а також вказати всі помічені помилки або моменти, які викликали труднощі чи непорозуміння (рис. 3.6). Крім того,

було організовано спільне обговорення результатів, що дозволило зібрати цінні пропозиції стосовно покращення ігрового процесу.

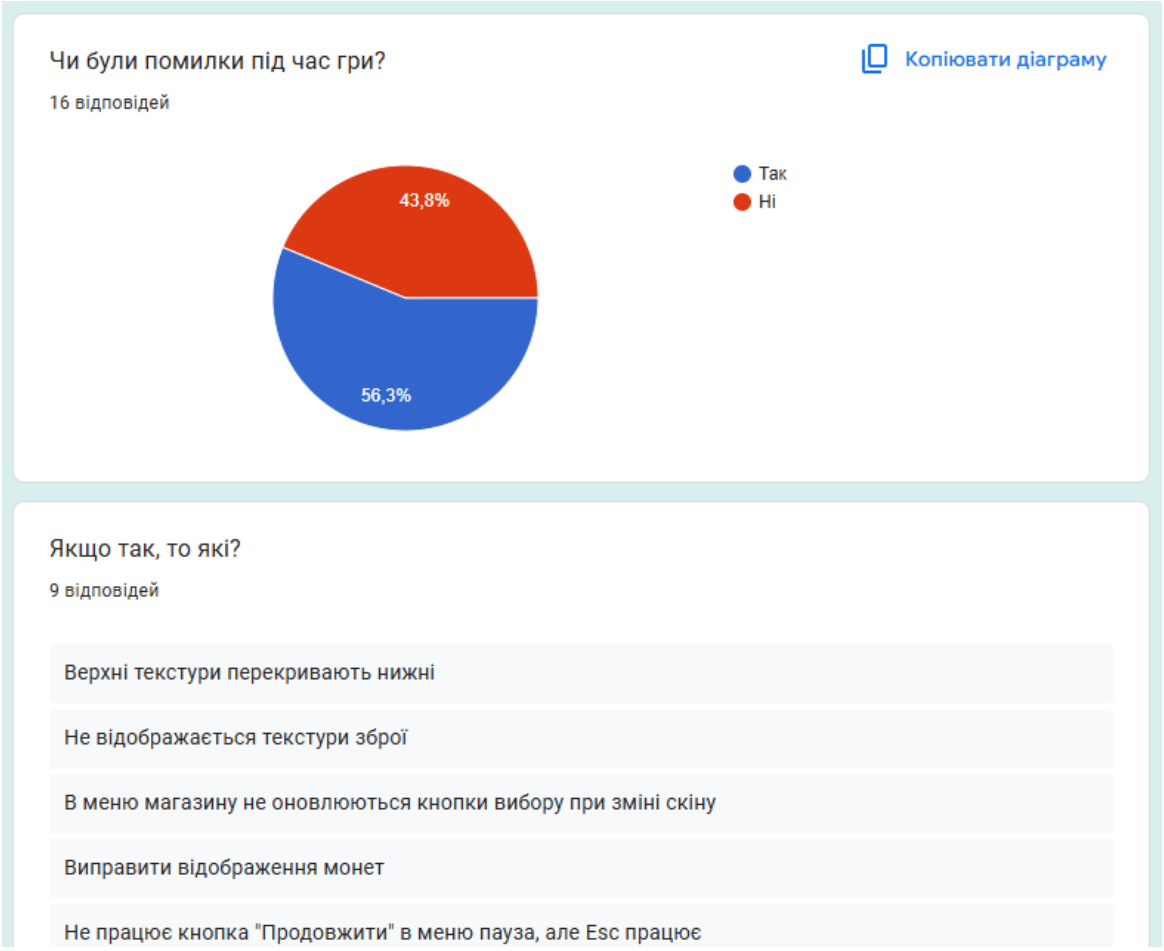

Рисунок 3.6 – Результати тестування

На основі зібраних відгуків було вирішено реалізувати систему вибору рівня складності, оскільки деякі гравці вважали гру надто легкою, тоді як інші не могли впоратись і на даному рівні. В результаті було додано три режими: легкий, середній та складний, які змінюють кількість здоров’я ворогів, їхню шкоду та інші аспекти.

Також після побажань тестувальників у грі була розширена система кастомізації персонажа – реалізовано можливість змінювати зовнішній вигляд гравця в окремому меню. Це не лише дозволяє персоналізувати ігровий досвід, але й додає візуального різноманіття.

Ще одним важливим удосконаленням стала доопрацьована бойова система. Було оптимізовано анімацію стрільби, покращено відчуття влучання по ворогу, додано візуальні ефекти при завданні шкоди, а також відкориговано баланс між боєм з ворогами. Це значно підвищило динаміку та задоволення від боїв у грі.

Контрольна група також проводила тестування інсталяційного пакета на різних комп’ютерах із різними характеристиками та роздільною здатністю екрана. Це дозволило виявити потенційні проблеми з сумісністю, відсутніми бібліотеками або конфігураціями, які могли вплинути на працездатність гри. За результатами тестування були внесені корективи до інсталяційного процесу, зокрема уточнено перелік необхідних компонентів, покращено обробку помилок і забезпечено правильне розгортання гри на більшості систем.

3.3 Розробка графічного контенту гри

У процесі практичної реалізації гри особлива увага була приділена створенню візуального стилю, який відповідає загальній концепції світу у піксельному виконанні. Зважаючи на специфіку проєкту, більшість графічних елементів було вирішено створити власноруч. Це дало змогу контролювати якість візуального контенту та забезпечити єдність стилю на всіх рівнях гри.

Процес створення піксельної графіки для гри здійснювався за допомогою спеціалізованої програми Aseprite, яка є популярним інструментом серед художників піксель-арту. Програма має зручний інтерфейс, що дозволяє легко працювати з шарами, анімацією та палітрою кольорів. Завдяки простому управлінню, можна швидко створювати як окремі спрайти, так і повноцінні анімаційні цикли.

У ході роботи використовувалися базові інструменти малювання: олівець, гумка, заливка та інструменти для симетрії, що значно прискорювали процес створення повторюваних елементів (рис. 3.7). Зокрема, для анімацій передбачено

таймлайн, де можна зручно перемикати та редагувати кадри. Це дозволяє в режимі реального часу бачити, як виглядатиме рух чи дія у грі.

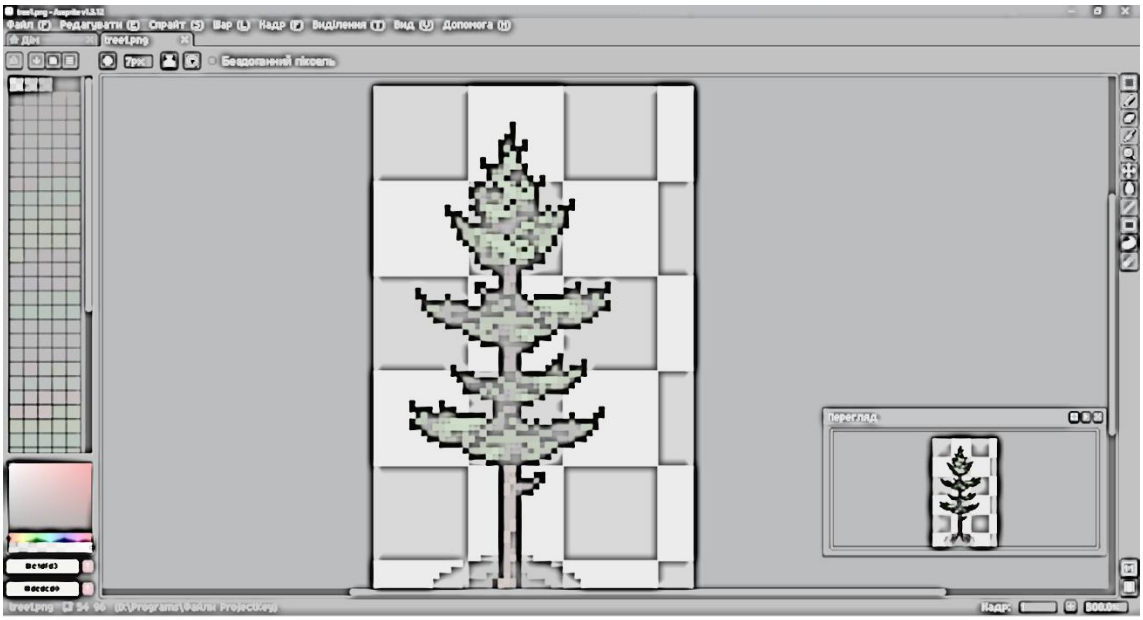

Рисунок 3.7 – Процес створення спрайту дерева

За допомогою графічного редактору було створено десятки унікальних графічних елементів, серед яких – текстури головного героя у різних анімаційних станах таких як: рух, стрільба та отримання шкоди, а також відповідні анімації для ворогів.

Окрему увагу було приділено оточенню. Промальовано дерева, будівлі, кущі та травинки, які згодом були згруповані в масиви для зручної генерації рівнів. Кожен з цих об’єктів був адаптований до відповідного біому – зимовий або літній варіанти дерев, що дозволяє рівням виглядати більш різноманітно й природно. Стилістично елементи поєднують простоту форм і приглушену палітру, що дозволяє акцентувати увагу на динаміці боїв та взаємодії з ворогами.

Попри наявність великої кількості готових безкоштовних ресурсів у Asset Store, практично кожен з них проходив ретельне редагування перед інтеграцією у гру. Часто доводилось змінювати колір, контури, деталізацію або зовсім перемальовувати текстури, щоб вони не порушували загальну стилістику гри.

Готові ресурси використовувались лише у випадках, коли це було виправдано з точки зору економії часу і при цьому не шкодило візуальній єдності.

Крім елементів ігрового світу, я створив елементи інтерфейсу: панелі, іконки, а також кнопки головного меню, налаштувань та магазину (рис. 3.8). Ці елементи були розроблені з урахуванням зручності взаємодії для гравця та загальної стилістики гри.

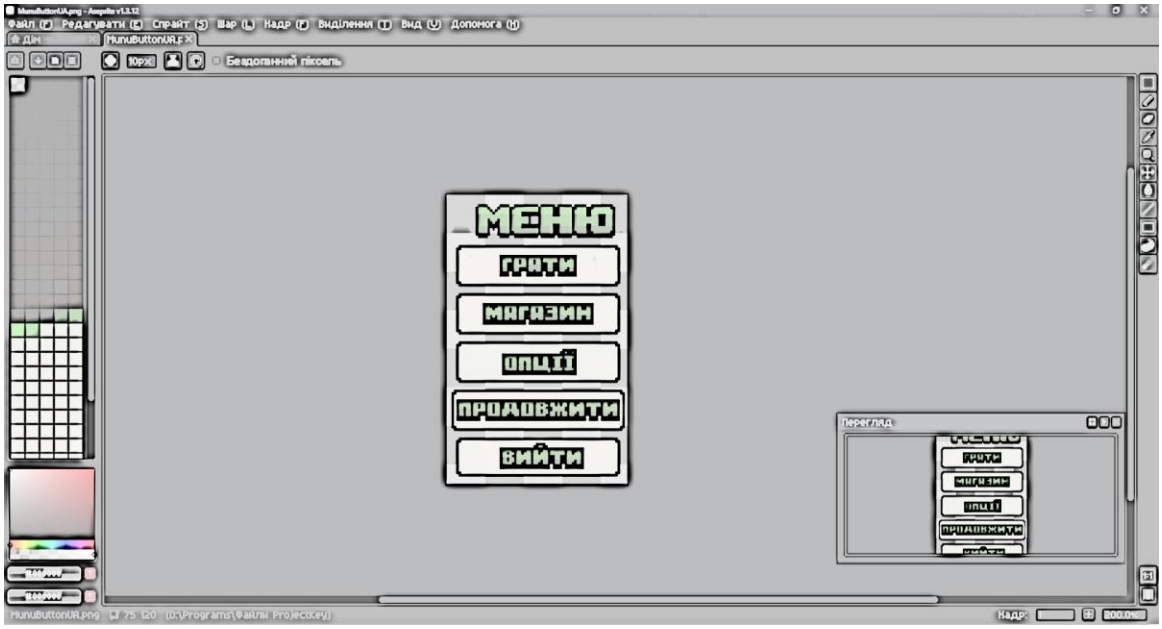

Рисунок 3.8 – Кнопки головного меню

Після створення графічних елементів, наступним важливим етапом стало їх правильне інтегрування в Unity. Щоб спрайти коректно відображались у грі, їх необхідно було імпортувати у форматі PNG для збереження прозорого фону, після чого в Unity виконувались додаткові налаштування.

Насамперед, при імпорті великих спрайт-листів, де було зображено більше одного елементу, потрібно було розрізати їх на окремі кадри за допомогою Sprite Editor. Для цього використовувалась сітка з фіксованим розміром кадру або ручне розбиття, якщо розміри спрайтів відрізнялись. Це дозволяло створити анімації, які згодом об'єднувались в Animator Controller для кожного об'єкта.

Дуже важливою частиною цього процесу є встановлення точки орієнтації (pivot). Вона визначає, навколо якої точки об’єкт буде обертатись,

масштабуватись і позиціонуватись у сцені. Наприклад, для персонажів pivot зазвичай ставився у нижній частині тіла, тоді як для зброї – ближче до місця тримання. Неправильно встановлений pivot міг призвести до зміщення під час анімації, накладання текстур в не правильному порядку або некоректного напрямку стрільби, тому цей етап потребував уважності.

Після налаштувань і розмітки, текстури використовувались для створення префабів. Кожен префаб містив не лише зображення, а й фізичні компоненти, скрипти взаємодії та анімаційні компоненти. Завдяки правильній підготовці графіки в редакторі та точній інтеграції в Unity, вдалось досягти якісного та стабільного візуального результату в грі.

Таким чином, створення графічного контенту стало однією з найважливіших і найбільш трудомістких частин проєкту. Проте завдяки цьому вдалося досягти цілісного художнього стилю, що підкреслює атмосферу гри та робить її візуально привабливою й впізнаваною. Aseprite став ефективним інструментом у створенні унікальної візуальної частини гри, дозволив зберегти цілісність художнього стилю та забезпечити якісну піксельну графіку, яка відповідає сучасним тенденціям у 2D інді-розробці.

3.4 Розробка документації для програмного продукту

Реалізація внутрішньої документації для гравця через кнопку у грі потребує створення зручного та інтуїтивного інтерфейсу, який дозволить ознайомитися з основною інформацією про гру. Такий функціонал є важливою частиною користувацького досвіду, оскільки не кожен гравець готовий витрачати час на вивчення зовнішніх інструкцій чи туторіалів. Замість цього, все необхідне повинно бути доступне прямо в грі, у зрозумілому вигляді.

У грі реалізовано спеціальну кнопку з іконкою знака питання, натискання на яку відкриває окреме вікно допомоги. Це вікно побудоване за допомогою UI-системи Unity. Всередині цього вікна розміщено основну текстову інформацію,

поділену на логічні блоки: управління, пояснення інтерфейсу, цілі гри, механіки та іншу важливу інформацію (рис. 3.9). Інтерфейс реалізований за допомогою вертикального списку, що дозволяє легко орієнтуватися в інформації. Уся система допомоги працює всередині одного Canvas, який вмикається та вимикається під час гри.

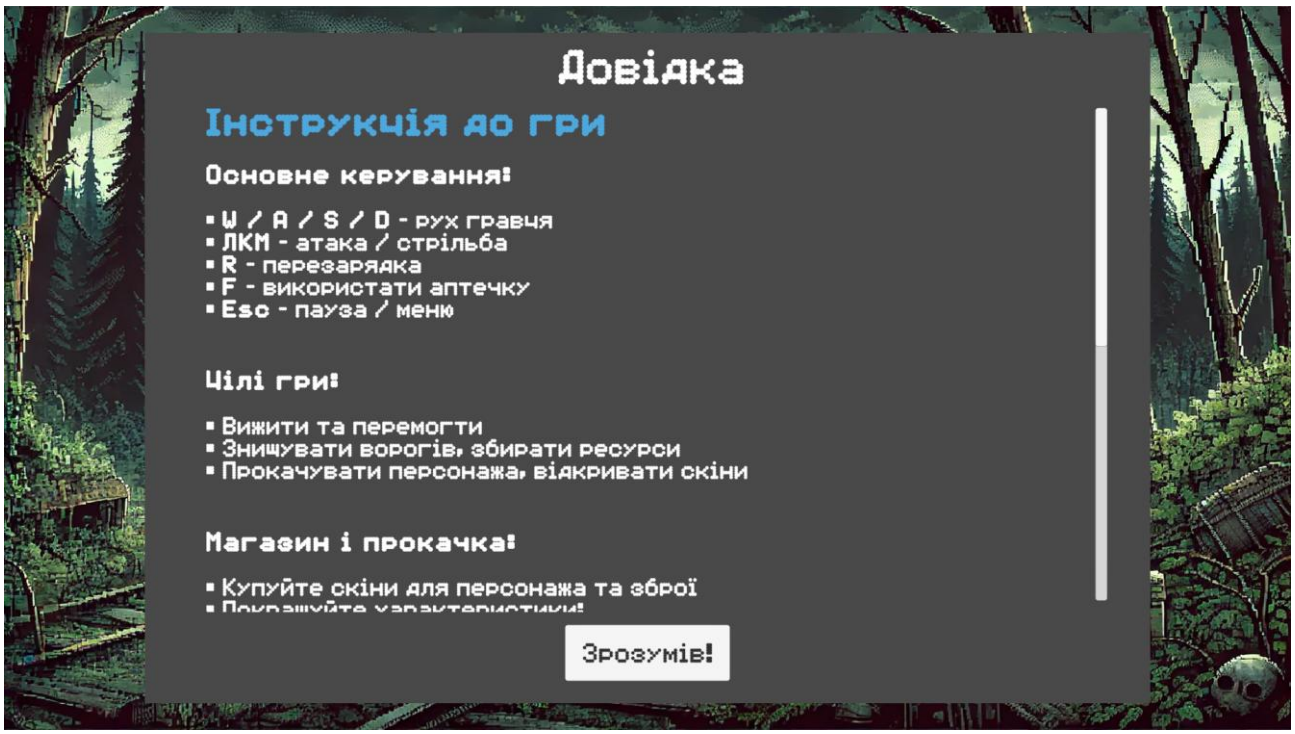

Рисунок 3.9 – Вікно з довідкою для користувача

В цілому, вбудована довідка надає гравцю простий і доступний інструмент для ознайомлення з основними аспектами гри прямо в процесі проходження, без відволікання на зовнішні джерела. Це підвищує комфорт, знижує поріг входу новачкам і загалом позитивно впливає на враження від гри.

3.5 Розробка інсталяційного пакета

Після завершення розробки гри важливо забезпечити зручне встановлення кінцевого продукту для користувачів. З цією метою було вирішено створити

інсталяційний пакет, який дозволяє просто і швидко встановити гру на комп’ютер. Для цього було обрано інструмент Inno Setup, який є одним із найпопулярніших безкоштовних рішень для створення інсталяційних програм під Windows.

Inno Setup дозволяє створити настроюваний інсталяційний файл з графічним інтерфейсом, що автоматизує копіювання файлів гри в обрану директорію, створення ярликів на робочому столі, а також запис необхідних параметрів у реєстр або системні файли, якщо це потрібно.

У процесі створення інсталятора було підготовлено всі необхідні файли гри, включно з виконуваним файлом, ресурсами, тобто графіка, сцени, налаштування, також бібліотеками і збереженими параметрами користувача. Створено конфігураційний скрипт Inno Setup, де було визначено шлях встановлення за замовчуванням, ім’я програми, ярлики, а також умови деінсталяції.

Після завершення налаштувань було згенеровано інсталяційний файл, який перевірено на кількох комп’ютерах із різними конфігураціями. Встановлення проходило успішно: усі файли були розпаковані до заданої директорії, ярлики створювались коректно, а гра запускалась без помилок. Процес інсталяції показано на рисунку 3.10.

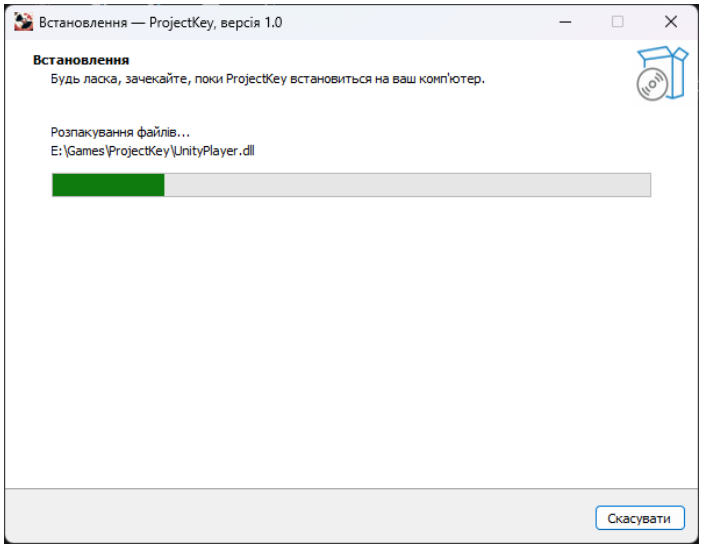

Рисунок 3.10 – Процес встановлення

Завдяки використанню Inno Setup, підготовлений інсталяційний файл має компактний розмір, не вимагає додаткових інструментів на стороні користувача, і дозволяє легко розповсюджувати гру через файлообмінники або зовнішні платформи. Це значно спрощує поширення гри серед користувачів, особливо в умовах незалежної розробки.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було досягнуто основної мети – розробити 2D гру в піксельному стилі з генерацією рівнів на базі ігрового рушія Unity. Проєкт реалізує основні елементи ігрової механіки, візуального оформлення та логіки генерації рівнів.

Було проведено аналіз предметної області, у якому розглянуто сучасні алгоритми процедурної генерації, зокрема шумові функції та методи розміщення об’єктів. Також досліджено актуальність піксельної графіки, що, попри технічний прогрес, залишається популярною серед інді-розробників і гравців, як і загальний інтерес до 2D ігор у ретро-стилі.

Визначено функціональні та нефункціональні вимоги до гри, які стали основою для розробки. З метою моделювання взаємодії користувача із системою була створена UML-діаграма варіантів використання, що наочно демонструє основні сценарії поведінки гравця.

Було обрано інструменти реалізації проєкту: рушій Unity як основна платформа розробки, алгоритм генерації на основі шуму Перліна, система AI із використанням NavMesh для навігації, графічний редактор Aseprite для створення спрайтів, GitHub для контролю версій та Trello для організації робочого процесу.

У результаті реалізовано повноцінний прототип гри, що включає генерацію рівнів, бойову систему, базову економіку та механіку апгрейду, виконаний відповідно до сформованих вимог.

Було проведено функціональне та користувацьке тестування гри, виявлено низку недоліків, які були усунені. Оптимізовано логіку зіткнень та взаємодію об’єктів для забезпечення стабільної роботи гри на різних пристроях.

Створено повний набір графічного контенту, включно з персонажами, ворогами, об’єктами оточення та елементами інтерфейсу. Більшість текстур

створено вручну в Aseprite, решта адаптована з відкритих ресурсів під загальну стилістику.

Підготовлено документацію до гри, що включає пояснення ігрових елементів, управління, опис механік і налаштувань. Інформація реалізована у вигляді інтерактивного меню довідки в грі.

Розроблено інсталяційний пакет із використанням Inno Setup. Проведено успішне тестування встановлення гри на різних ПК, що підтверджує працездатність інсталятора та його зручність для кінцевого користувача.

Таким чином, у результаті виконаної кваліфікаційної роботи було не лише створено повноцінний ігровий продукт, але й опрацьовано повний цикл розробки – від проєктування до тестування та підготовки до публікації, що відповідає сучасним вимогам до інженерії програмного забезпечення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Adobe photoshop. Adobe: Creative, marketing and document management solutions. URL: https://www.adobe.com/ua/products/photoshop.html (date of access: 17.03.2025).

2. A* pathfinding project. Arongranberg.com – Game Development with Unity 3D. URL: https://arongranberg.com/astar/ (date of access: 08.03.2025).

3. Aseprite. Aseprite - Animated sprite editor & pixel art tool. URL: https://www.aseprite.org/ (date of access: 18.03.2025).

4. Best approach for steering behaviors implementation in dots. Unity Discussions. URL: https://discussions.unity.com/t/best-approach-for-steering-behaviors-implementation-in-dots/790919 (date of access: 10.03.2025).

5. Blog - yvan scher. home - yvan scher. URL: https://yvanscher.com/writing/playing_with_perlin_noise (date of access: 06.02.2025).

6. C# - a modern, open-source programming language .NET. Microsoft. URL: https://dotnet.microsoft.com/en-us/languages/csharp (date of access: 04.03.2025).

7. Dead Cells вики. Dead Cells Wiki. URL: https://dead-cells.fandom.com/ua/wiki/ (дата звернення: 15.02.2025).

8. GitHub · Build and ship software platform. GitHub. URL: https://github.com/ (date of access: 13.03.2025).

9. Godot Engine - Free and open source 2D and 3D game engine. Godot Engine. URL: https://godotengine.org/ (date of access: 24.05.2025).

10. Nsdg. Вступ до процедурної генерації в Unity. Sharp Coder Blog. URL: https://uk.sharpcoderblog.com/blog/an-introduction-to-procedural-generation-in-unity (дата звернення: 04.02.2025).

11. Pixel-art. Gamedev.dou.ua. URL: https://gamedev.dou.ua/articles/c-games-revolution/ (date of access: 01.05.2025).

12. Spelunky world. URL: https://www.spelunkyworld.com/ (date of access: 17.02.2025).

13. Steam store. Welcome to Steam. URL: https://store.steampowered.com/ (date of access: 09.02.2025).

14. The binding of isaac wiki. URL: https://bindingofisaac.fandom.com/wiki/ (date of access: 13.02.2025).

15. Trello. Capture, organize, and tackle your to-dos from anywhere. URL: https://trello.com/home (date of access: 20.03.2025).

16. Unity - mathf.perlinnoise. Unity - Manual: Unity 6.1 User Manual. URL: https://docs.unity3d.com/en/530/ScriptReference/Mathf.PerlinNoise.html (date of access: 07.02.2025).

17. Unity - Scripting API: NavMesh. Unity - Manual: Unity 6.1 User Manual. URL: https://docs.unity3d.com/ScriptReference/AI.NavMesh.html (date of access: 07.03.2025).

18. Unreal engine course. N-iX Games. URL: https://gamestudio.n-ix.com/unreal-engine-course/ (date of access: 02.03.2025).

ДОДАТКИ

Додаток А

Тези X Міжнародної науково-практичної конференції з проблем вищої освіти і науки «Інформаційні технології в освіті, науці і виробництві (ІТОНВ-2025)» 23-24 травня 2025 р. Луцьк 2025

Міністерство освіти і науки України
Волинська обласна рада
Луцька міська рада
Луцький національний технічний університет
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського»
Національний університет «Львівська політехніка»
Військовий інститут телекомунікацій та інформатизації імені Героїв Крут (м. Київ)
Тернопільський національний педагогічний університет імені В. Гнатюка
Рівненський державний гуманітарний університет
Навчально-науковий інститут «Українська інженерно-педагогічна академія» Харківського національного університету ім. В.Н. Каразіна
Люблінська політехніка (Польща)
Фрайберзька гірнича академія (Німеччина)
Гронінгенський університет (Нідерланди)
Кавказький університет (Грузія)
Політехнічний університет Браганси (Португалія)

ІТОНВ

ТЕЗИ ДОПОВІДЕЙ
X Міжнародної науково-практичної конференції з проблем вищої освіти і науки «Інформаційні технології в освіті, науці і виробництві (ІТОНВ-2025)

23-24 травня 2025 р.

Луцьк 2025

СЕКЦІЯ 5. ІНФОРМАЦІЙНІ ТА ІНТЕЛЕКТУАЛЬНІ СИСТЕМИ І ТЕХНОЛОГІЇ		
Антонюк А.О. **Повстяна Ю.С.**	Голосовий помічник на основі штучного інтелекту	275
Вовк П.Б.	Підготовка середовища та налаштування локального кластера Kubernetes: від теорії до практики	282
Вознюк А.В. **Данилко К.С.**	Розробка 2D ігор на unity з генерацією світу	286
Вознюк М.В. **Неділько О.В.**	Аналіз впливу темної теми на зручність користування веб-додатками	289
Вознюк А.В. **Хітько Ю.В.**	Автоматизація сервісів доставки їжі за допомогою Telegram-бота з AI та платіжним модулем	293
Вознюк А.В. **Чоботар Ю.С.** **Примич М.О.**	Технічні рішення та освітні переваги при створенні і впровадженні вебплатформи для онлайн-навчання	298
Газдюк К.П. **Дмитращук К.М.** **Кривинчук В.О.**	Адаптивне планування логістичних маршрутів в умовах воєнного стану	304
Головня С.А.	Застосування сучасних бібліотек Python для задач з аналізу та прогнозування	308
Демідов П.Г. **Андрієнко В.А.** **Маргаза Д.Ю.** **Муляр В.П.**	Порівняльна характеристика нейро-нечітких та нейронних технологій на прикладі розв'язання задачі прогнозування вартості валюти на фінансовому ринку	312

УДК 004.514

РОЗРОБКА 2D ІГОР НА UNITY З ГЕНЕРАЦІЄЮ СВІТУ

Вознюк Анастасія Вадимівна
Луцький національний технічний університет, асистент кафедри інженерії програмного забезпечення, a.vozniuk@lutsk-ntu.com.ua
Данилко Костянтин Сергійович
Луцький національний технічний університет, студент 4 курсу кафедри інженерії програмного забезпечення, danilkokosta@gmail.com

2D GAME DEVELOPMENT IN UNITY WITH WORLD GENERATION

Vozniuk Anastasiia Vadymivna
Lutsk National Technical University, Assistant at the Department of Software Engineering, a.vozniuk@lutsk-ntu.com.ua
Danylko Kostiantyn Sergiyevich
Lutsk National Technical University, Student at the Department of Software Engineering, danilkokosta@gmail.com

The features of creating two-dimensional computer games using the Unity engine and methods of generating the game world are presented. The advantages of using procedural generation in modern indie games, its impact on replayability and reducing the cost of content production are noted. Architectural approaches to building game levels, the use of noise algorithms and the advantages of modular development using Tilemap and ScriptableObject are analyzed.
***Keywords**: Unity, 2D game, procedural generation, C#, Tilemap, game design, indie development.*

Стрімке зростання популярності відеоігор, особливо інді-проєктів, зумовлює потребу у технологіях, що дозволяють створювати масштабні та варіативні ігрові світи без суттєвих фінансових затрат. Одним із таких підходів є процедурна

генерація контенту — технологія, що дозволяє автоматично створювати рівні, середовища, персонажів або об'єкти під час виконання гри.

Unity, як одна з найпопулярніших платформ для розробки 2D-ігор, надає розробникам гнучкі інструменти для реалізації процедурної генерації. У поєднанні з мовою програмування C# та вбудованими можливостями, такими як Tilemap, Physics2D, ScriptableObject та NavMesh, Unity дозволяє створювати динамічні й варіативні світи з мінімальними витратами на ручну розробку кожного рівня.

Шум Перліна є одним із найпоширеніших інструментів для створення природного вигляду ландшафтів. Він дозволяє отримати псевдовипадкове значення в кожній точці координатної сітки, яке плавно змінюється, що імітує природну нерівність поверхні. Це значення може використовуватись для визначення висоти місцевості або вибору типу плитки, яка буде розміщена в конкретному місці карти.

Клітинні автомати широко застосовуються для генерації рівнів через свою простоту й гнучкість. Суть методу полягає в тому, що кожна клітинка на сітці має початковий стан, після чого до неї послідовно застосовуються правила переходу до наступного стану, які залежать від її сусідів. Завдяки цьому формується складна, але керована структура — наприклад, печери, лабіринти або приміщення.

Генетичні алгоритми базуються на принципах природної еволюції та використовуються для пошуку оптимальних рішень у складних завданнях. У контексті генерації рівнів вони дозволяють створювати велику кількість варіантів ігрових карт, які згодом відбираються та вдосконалюються на основі заданих критеріїв якості — наприклад, прохідності, балансу складності або кількості ресурсів.

Успішна реалізація генерації світу у 2D-іграх вимагає балансування випадковості та контрольованої структури. Надто хаотична генерація може негативно вплинути на геймплей, тоді як надто передбачувана — знижує інтерес до гри. Для

досягнення балансу використовуються попередньо підготовлені модулі, які комбінуються у випадковому порядку. Генерація за шаблонами, де зберігається загальна логіка рівня, але наповнення змінюється. Семантична генерація, яка враховує функціональність ігрових зон.

Інтеграція процедурної генерації дозволяє також ефективно реалізовувати системи прогресії, навігації, випадкових подій та динамічного середовища. Багато успішних інді-ігор, зробили ставку саме на змінний ігровий світ, що стимулює повторне проходження гри.

Ігровий процес повинен бути оптимізованим — зайве навантаження на систему, особливо під час генерації рівнів, може призводити до втрати кадрів або зависань. Для цього використовуються обчислення поза головним потоком або попередня генерація частин світу.

Робота з анімацією у 2D в Unity часто базується на анімаційних контролерах, які дозволяють створювати плавні переходи між станами — біг, атака, отримання шкоди тощо. Це суттєво покращує візуальне сприйняття гри.

Реалізація штучного інтелекту ворогів дозволяє зробити гру цікавішою — найпростіші варіанти включають патрулювання, переслідування гравця, уникнення перешкод. Поведінка ворогів часто програмується через стан-машини.

Таким чином, розробка 2D-ігор із процедурною генерацією в середовищі Unity є перспективним напрямом, що поєднує гнучкість технологій, креативність дизайну та ефективне використання ресурсів. Такі ігри мають потенціал для виходу на ринок як комерційний продукт, з можливістю подальшого масштабування, розширення функціоналу, підтримки модифікацій або багатокористувацької гри.

Список використаних джерел

1. Unity Technologies. *Unity Manual – 2D Game Development*. URL: https://docs.unity3d.com/Manual/2D.html (дата звернення: 02.05.2025).

2. Unity Learn. *Create a Procedurally Generated Dungeon*. URL: https://learn.unity.com/tutorial/procedural-cave-generation-tutorial (дата звернення: 03.05.2025).
3. Official Unity Tutorials. URL: https://www.youtube.com/user/Unity3D (дата звернення: 04.05.2025).
4. Catlike Coding. *Noise-based Map Generation in Unity*. URL: https://catlikecoding.com/unity/tutorials/ (дата звернення: 04.05.2025).

УДК 004.42:004.738.5

АНАЛІЗ ВПЛИВУ ТЕМНОЇ ТЕМИ НА ЗРУЧНІСТЬ КОРИСТУВАННЯ ВЕБ-ДОДАТКАМИ

Вознюк Марія Віталіївна
Луцький національний технічний університет, здобувач вищої освіти II курсу, спеціальності F3 «Комп'ютерні науки», voznukmaria7@gmail.com
Неділько Ольга Володимирівна
Луцький національний технічний університет, асистент кафедри комп'ютерних наук

ANALYSIS OF DARK THEME IMPACT ON WEB APPLICATION USABILITY

Voznyuk Maria Vitalievna
Lutsk National Technical Universit, Student, Specialty F3 "Computer Science, voznukmaria7@gmail.com

This paper explores the effect of dark theme design on the usability of web applications. It analyzes aspects such as user comfort, visual strain, and accessibility in modern interfaces.

***Keywords**: dark theme, usability, accessibility, web interface, user experience.*

Темна тема інтерфейсу веб-додатків стала важливим елементом сучасного дизайну. Її популярність обумовлена як естетичними міркуваннями, так і практичними перевагами. Зокрема, використання темного фону може знижувати