Вищий навчальний заклад

«Університет економіки та права «КРОК»

Фаховий коледж

Циклова комісія з інформаційних технологій

Кваліфікаційна робота фахового молодшого бакалавра

на тему Створення 3D гри на Unity за мотивами легенди про «Гамельнського щуролова»

Виконав _____

(Підпис)

_____Героєв Кирил Максимович_____

(прізвище, ім'я, по батькові)

_____

Науковий керівник

_____Уваров Леонід Миколайович____

(прізвище, ім'я, по батькові)

_____

_____

(Резолюція «До захисту»)

Попередній захист:

_____

(Висновок: "До захисту в екзаменаційній комісії ")

Голова циклової комісії _____ _____ _____

(Підпис) (Прізвище, ініціали) (Дата)

Київ – 2025 року

ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД

УНІВЕРСИТЕТ ЕКОНОМІКИ ТА ПРАВА «КРОК»

Фаховий коледж

Циклова комісія з інформаційних технологій

Спеціальність 121 інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:

Голова циклової комісії _____ Леонід УВАРОВ

(підпис)

«____» _____ 2025 року

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Здобувач освіти Героєв Кирил Максимович

1. Тема роботи Створення 3D гри на Unity за мотивами легенди про «Гамельнського щуролова»

затверджена наказом по університету від «___» _____ 202__ р. № _____

2. Термін здачі закінченої роботи «30» травня 2025 року
3. Вихідні дані до роботи:

а) цільова аудиторія – вікова категорія гравців, на яку орієнтована гра (діти, підлітки, дорослі) та вимоги: просте управління та навігація, доступний та зрозумілий сюжет, елементи гри на розвиток логіки;

б) функціональність – розробити 3D гру від третьої особи, з зручим інтерфейсом, меню, пауза, карта, завдання збір ресурсів, спілкування з NPC, головоломки, приманювання щурів, реалізувати декілька рівні та локацій: місто лісова місцевість, звуковий супровід;

в) технічні вимоги –ігровий движок (Unity, Godot, GameMakerStudio2 тощо), мова програмування (C#, GDScript, GML тощо) вимоги до графіки та звукового супроводу, платформ, на яких буде доступна гра (Windows, macOS, Android, iOS тощо);

г) існуючі рішення (приклади ігор).

4. Зміст пояснювальної записки

а) Розділ 1 Теоретична частина. Аналіз існуючих рішень, обґрунтування функціональності, вибору платформи, технології та технічних вимог для розроблення гри. Опис легенди сюжетна основа.

б) Розділ 2 Проєктування та розробка. Створення цікавого та захоплюючого ігрового процесу, розробка візуального стилю гри та персонажів, сюжетна основа, створення програмного коду гри, реалізація ігрової механіки та інтерфейсу користувача, інтеграція графіки та звуку, забезпечення коректної роботи гри.

в) Розділ 3 Експериментальна частина. Перевірка роботи гри на різних пристроях та платформах, виявлення та виправлення помилок, балансування складності гри. Результати тестування. Інструкція користувачам (для технічного адміністратора – опис процесу розробки гри, її архітектури та функціональності, пояснення щодо використання обраного ігрового движка та інструментів розробки, для користувача – пояснення правил гри та принципів її роботи).

5. Перелік графічного матеріалу:

Скріншоти існуючих рішень

Скріншоти інтерфейсу продукту

Схеми і таблиці щодо візуалізації аналізу

Блок-схеми алгоритмів

Діаграми щодо проєктування продукту

Дата видачі завдання «12» лютого 2025 року

Науковий керівник _____ Леонід УВАРОВ

(підпис)

Завдання прийняв до виконання _____ Кирил ГЕРОЄВ

(підпис)

РЕФЕРАТ

Пояснювальна записка: 57 сторінка, 29 рисунків, 15 джерел.

Об'єкт дослідження – гра на основі легенди про «Гамельнського щуролова».

Мета роботи – розробка гри на основі механік, що дуже схожі на особливі вміння головного героя з легенди про «Гамельнського щуролова».

Кваліфікаційна робота містить результати розробки 3D гри на Unity з механіками контролю щурів з відстані та їх взаємодія з навколишнім середовищем. Проведено аналіз предметної області, розроблені варіанти взаємодії та їх умови, а також побудовані алгоритми для схематичного пояснення їх роботи.

Результати можуть бути впроваджені в навчальних закладах України.

РОЗРОБКА МЕХАНІК, ПЛАНУВАННЯ ЛАНДШАФТУ, ВЗАЄМОДІЯ З ПЕРСОНАЖЕМ, АНІМАЦІЯ ПОДІЙ, НАЛАШТУВАННЯ ІНТЕРФЕЙСУ, ACET.

ABSTRACT

Explanatory note: 57 pages, 29 figures, 15 sources.

The object of the research is a game based on the legend of the "Pirate of Hamelin".

The purpose of the work is to develop a game based on mechanics that are very similar to the special abilities of the main character from the legend of the "Pied Piper of Hamelin".

The explanatory note of the qualification work contains the results of the development of a 3D game on Unity with the mechanics of controlling rats from a distance and their interaction with the environment. An analysis of the subject area was conducted, interaction options and their conditions were developed, and algorithms were built to schematically explain their work.

The results can be implemented in educational institutions of Ukraine.

DEVELOPMENT OF MECHANICS, LANDSCAPE PLANNING, INTERACTION WITH THE CHARACTER, ANIMATION OF EVENTS, INTERFACE CUSTOMIZATION, ASSET.

ЗМІСТ

Скорочення та умовні познаки

ШІ	– штучний інтелект
WEB	– world wide web
ПК	– персональний комп’ютер
FPS	– Frames Per Second
2D	– 2 Dimensional
3D	– 3 Dimensional
UI	– User Interface
Prefab	– Prefabricated Object
FBX	– Формат 3D-моделей
API	– Application Programming Interface

ВСТУП

Актуальність завдання (задачі): Коли нам дали на проект вибір між сайтами, іграми та прогресивними ботами в Telegram, я одразу вирішив, що буду робити саме гру. Маючи вже певний досвід у розробці WEB-сайтів, зрозумів що мені буде просто не цікаво займатись цим, тому вибір був між іграми там чат-ботом. Але як для мене, другий варіант, не виглядає чимось серйозним, що справді могло б дати важливі навички та досвід, за умови, що я захочу в майбутньому продовжити працювати в цьому напрямку.

В результаті, я зупинився саме на грі та почав шукати, які саме додатки та двигуни можуть стати основою. Інші причини вибору саме гри, між іншими варіантами – це постійна популяризація розробки інді-ігор (ігри, які створенні повністю однією людиною з повного нуля). Доказом цього, можна вважати те, що з кожним роком, все частіше нагороди за найкращі ігри року, забирають саме ігри такого типу, а між геймерами існує думка, що у сучасному світі, тільки у таких проектів є хоч якась душа (не просто спосіб заробити більше грошей).

Також причинами популяризації саме інді-розробки стає швидкий розвиток штучного інтелекту, що може стати в нагоді, полегшити та прискорити розробку великих ігор. І навіть якщо на даний момент ШІ не можуть замінити програмістів (часто пишуть з помилками, інколи ледь виконують найлегші завдання для програмістів), вони все ще продовжують розвиватись, оновлюватись. Це дає надію на значний прогрес в технологіях не тільки звичайним програмістам, а й великим ІТ-компаніям. А навіть зараз вже їх починають використовувати для банальних завдань, що обов’язково повинні бути виконані, але при цьому доволі легкі у виконанні.

Замінюючи в таких задачах людей на ШІ, ми економимо час, який можна буде використати в подальшому для виконання інших, важливіших та більш креативних завдань.

Запитавши в ШІ про популярні та актуальні програми я отримав такі результати : Godot (рис. 1.1), Unity (рис. 1.3) та Unreal Engine 5 (рис. 1.2).

Провівши певні додаткові дослідження, що були спрямовані на порівняння між собою наведених вище прикладів, зміг виділити головні моменти кожного з них. Треба зауважити, що інформацію я черпав не тільки з книг та статей, а й з відео з інтернету. Головна причина цього - постійне оновлення відео на новіші, в той час як книги завжди старішають. Якщо до цього ще піднести те, що в наш час технології розвиваються з шаленою швидкістю, думати що через книги справді можна знайти актуальну та нову інформацію – смішно та майже неможливо. З таких відео я зробив декілька висновків, що описують головні характеристики таких двигунів.

Godot – це безкоштовний ігровий рушій з відкритим кодом, який був створений у 2007 році як внутрішній інструмент для розробки ігор. Спочатку він використовувався невеликою командою, але згодом став доступним для всіх бажаючих. Godot відомий своєю простотою та гнучкістю, особливо завдяки системі вузлів, яка дозволяє легко структурувати гру. Він підтримує як 2D, так і 3D-розробку, а основним мовним інструментом є GDScript – мова, схожа на Python, що робить її зручною для новачків. Однак, незважаючи на свою легкість і доступність, Godot має менше готових асетів у порівнянні з конкурентами, а його інструменти для складних 3D-проектів ще не досягли рівня Unity чи Unreal Engine.

Unity – один із найпопулярніших ігрових рушіїв, який з’явився у 2005 році і швидко завоював аудиторію завдяки своїй простоті та універсальності. Він ідеально підходить для розробників різного рівня – від початківців до професіоналів. Unity дозволяє створювати ігри для різних платформ, включаючи ПК, мобільні пристрої та VR, а його Asset Store надає величезну кількість готових рішень. Основна мова програмування – C#, що робить його зручним для тих, хто вже має досвід у розробці. Однак деякі розробники

відзначають, що для досягнення високоякісної графіки часто потрібні додаткові плагіни, а зміни в ліцензійній політиці іноді викликають суперечки.

Unreal Engine 5 – це потужний рушій, який використовується для створення ААА-ігор і кінематографічних ефектів. Він був розроблений Epic Games і вперше представлений у 1998 році, а п’ята версія, випущена у 2022 році, принесла революційні технології, такі як Nanite та Lumen. Unreal Engine дозволяє створювати фотореалістичну графіку з мінімальними зусиллями, а його система Blueprints дає можливість розробляти ігри без глибоких знань програмування. Однак він вимагає потужного обладнання, і його інтерфейс може бути складним для новачків. Крім того, на відміну від Godot, Unreal бере роялті з доходів розробників після певного порогу, що може бути невигідно для невеликих студій.

Останній варіант насправді є найпопулярнішим за свою реалістичну графіку, зручність у використанні та велику кількість асетів з інтернету, проте він відпав одразу, бо у мене немає сильного ПК. Без нього кількість FPS в грі буде дуже низька, а розробка буде займати куди більше часу, якщо взагалі буде можлива.

Рисунок 1.1

Рисунок 1.2

Я обрав Unity, бо він є найпопулярнішим для розробки невеликих ігор, що не мають високих системних потреб та не займають багато місця на диску. Тим паче, я вже мав деякий досвід у програмуванні на C# та наявність вдома книг за цією мовою, яка в свою чергу є однією з найпопулярніших мов програмування зараз [10][9, с.3][12].

Рисунок 1.3

За основу для проекту я вирішив взяти легенду про «Гамельнського щуролова» (рис. 1.4). Причинами, саме такого, дивного на перший погляд варіанту, стали: потенціал реалізації та розвитку таких ідей як контроль щурів персонажем та керування ними, вічна актуальність розробки ігор на основі міфів та легенд різних країн та своя власна зацікавленість у цьому напрямку.

Рисунок 1.4

Для аналізу актуальних тем гри та програм для використання в розробці я використовував ШІ, знайдені в інтернеті статті, а також знайдені на Youtube відео по розробці та персональному досвіду в розробці таких проектів з повного нуля.

Мета роботи: Розробити гру, основні механіки якої, будуть нагадувати та, можливо, відтворювати легенду про «Гамельнського щуролова».

Завдання роботи: Розробити можливість контролювати щурів користувачем. Створити та налаштувати адаптивний UI-дизайн. Розробити мапу після ретельного планування ландшафту гри. Запровадити певні анімації переміщення та інших подій.

Об’єкт дослідження: 3D-гра від третього лиця на Unity створена за мотивами легенди про «Гамельнського щуролова».

Предмет дослідження: Часткове відтворення легенди про «Гамельнського щуролова» та реалізація основних властивостей головного героя.

Методи дослідження: Проведення аналізу основних функцій Unity, для початку роботи та ознайомлення з програмою. Спроби копіювати деякі елементи розробки з інтернет джерел. Навчання працювати з анімаціями

персонажа (анімації руху, зупинки на місці, контролю щурів), для подальшої імплементації в свій проект.

Практичне значення одержаних результатів: В результаті написання цієї кваліфікаційної роботи я навчусь працювати з Unity та анімаціями, що в майбутньому може стати мені в нагоді. Також не треба забувати, що цей проект також можна використати в резюме як приклад мого досвіду роботи з Unity.

Структура роботи: 57 сторінок, 29 рисунків, 15 посилань.

Ключові слова: РОЗРОБКА МЕХАНІК, ПЛАНУВАННЯ ЛАНДШАФТУ, ВЗАЄМОДІЯ З ПЕРСОНАЖЕМ, АНІМАЦІЯ ПОДІЙ, НАЛАШТУВАННЯ ІНТЕРФЕЙСУ, ACET.

РОЗДІЛ 1 ТЕОРЕТИЧНІ ОСНОВИ

1.1. Огляд літератури

Легенда про «Гамельнського Щуролова», яка походить із середньовічних європейських хронік, за століття свого існування набула численних інтерпретацій у літературі, мистецтві та сучасних медіа.

Аналіз наукових джерел свідчить, що цей сюжет традиційно розглядається через призму кількох ключових аспектів. Фольклористичні дослідження, зокрема праці братів Грімм та аналітичні роботи сучасних міфологів, акцентують увагу на історичних передумовах легенди. Вони розглядають її як символічне відображення соціальних потрясінь середньовіччя, пов’язуючи з епідеміями чуми або масовими міграціями.

Літературні приклади, такі як поема Йоганна Вольфганга Гете та сучасні художні твори, демонструють еволюцію образу Щуролова – від зловісного чаклуна до амбівалентного персонажа, що постає то як каратель, то як жертва обставин. У сфері цифрових медіа ця тема знайшла відображення переважно в нішевих проєктах, що дозволяє говорити про недостатнє її освоєння в ігровій індустрії. Проте аналіз схожих за тематикою ігор (наприклад, Bloodborne з його містичними мотивами чи A Plague Tale: Innocence (рис. 1.5) з акцентом на середньовічній атмосфері) свідчить про потенціал таких сюжетів для створення потужних ігрових всесвітів. Також не треба забувати, що навіть якщо не знаєте цю легенду досконально, я впевнений що Ви вже чули про цю легенду від інших, в якості стислого переказу сюжету або хоча б тільки кінцівку цієї історії. Це можна назвати гарним гарантом того, що користувачу (гравцю) пізніше не треба буде детально пояснювати механіки або чому все вийшло саме так, як воно побудовано в грі.

Рисунок 1.5

Теоретична база дослідження ґрунтується на кількох методологічних підходів. Літературно-історичний аналіз дозволив виявити ключові напрямки структури легенди та їх трансформацію в різних формах серед різних країн та народів. Цей підхід показав, що сюжет має значний потенціал для адаптації, оскільки поєднує універсальні архетипи з конкретними історичними контекстами.

З точки зору гейм-дизайну були розглянуті механіки, здатні ефективно передати унікальність легенди. Досвід таких ігор, як Hellblade: Senua's Sacrifice (рис. 1.6) у сфері психологічного напруження чи The Legend of Zelda: Ocarina of Time (рис. 1.7) у використанні музичних елементів, доводить можливість органічного поєднання фольклорної основи з сучасними ігровими системами.

Рисунок 1.6

Рисунок 1.7

Комплексний аналіз джерел дозволив виокремити кілька ключових напрямів для подальшої розробки гри. По-перше, було підтверджено, що легенда про Щуролова має значний адаптаційний потенціал завдяки своїй багатогранності – вона може бути основою як для містичного трилера, так і для філософської притчі. По-друге, дослідження показало ефективність використання музики як центрального ігрового механізму, що відповідає оригінальному сюжету. Це відкриває можливості для створення унікальних головоломок та системи взаємодії зі світом. По-третє, аналіз візуальних стилів схожих проєктів довів доцільність поєднання середньовічної естетики з елементами темного фентезі. Такий підхід дозволить одночасно передати історичний колорит і створити незабутню атмосферу. Таким чином, теоретичне дослідження підтвердило актуальність і перспективність обраної тематики, а також надало чіткі орієнтири для її втілення в ігровому середовищі.

Отримані результати стануть основою для подальшого проектування гри, забезпечуючи її глибину, оригінальність та популярність в подальшому.

1.2 Аналіз існуючих рішень

Легенда про «Гамельнського Щуролова», незважаючи на свою багатовікову історію, залишається відносно недослідженою територією в контексті сучасних відеоігор. Провівши ретельний аналіз існуючих інтерпретацій цього сюжету в різних медіа, можна виявити як унікальні можливості, так і значні прогалини, які ми плануємо заповнити нашим проектом. У літературній традиції, починаючи від братів Грімм до сучасних авторів, легенда отримала численні трактування, кожне з яких акцентує увагу на різних аспектах історії. Класичні інтерпретації, такі як поема Гете, зосереджені на моральній дилемі та символічному значенні образу Щуролова.

Сучасні ж автори часто розглядають цю легенду з точки зору психології, соціальних явищ або навіть політичної алегорії. Однак усі ці твори мають спільний недолік - відсутність інтерактивності, що обмежує можливості глибокого занурення в атмосферу та зміст легенди. У сфері кінематографа ситуація виглядає дещо краще. Фільми на кшталт " Щуролова " 2015 року демонструють спроби передати містичну атмосферу легенди, використовуючи сучасні технології та спеціальні ефекти. Проте навіть найкращі екранізації не можуть запропонувати глядачеві той рівень впливу на події та свободу дослідження світу, який є ключовим для відеоігор.

Що стосується ігрової індустрії, то тут ситуація виглядає найбільш проблематичною. Існуючі проекти, які так чи інакше торкаються теми Щуролова, можна розділити на дві категорії: застарілі текстовий пригоди (як "Pied Piper" 1986 року) та сучасні ігри, що лише використовують окремі мотиви легенди у своїх сюжетах ("A Plague Tale: Innocence").

Жоден з цих проектів не пропонує повноцінного досвіду, присвяченого саме цій легенді, не розкриває її багатогранність і не дозволяє гравцю по-справжньому зануритися в її світ. Саме ці прогалини та обмеження існуючих

рішень стали вирішальними факторами при розробці нашого власного підходу. Наша гра пропонує принципово новий рівень взаємодії з легендою, поєднуючи глибину літературних інтерпретацій, візуальну потужність сучасного кіно та унікальні можливості інтерактивного медіа.

Вибір рушія Unity для реалізації проекту був обумовлений цілим рядом ключових факторів. По-перше, крос-платформність Unity дозволяє нам охопити максимально широку аудиторію, включаючи гравців на ПК, консолях і мобільних пристроях. По-друге, потужний набір інструментів для 3D-розробки забезпечує всі необхідні можливості для створення багатого, деталізованого ігрового світу. Вбудована підтримка фізики, складної анімації персонажів і просунутих шейдерних ефектів дозволяє реалізувати навіть найамбітніші творчі задуми. Особливу цінність становить Unity Asset Store, який містить величезну кількість готових рішень - від середньовічних архітектурних наборів до спеціалізованих інструментів для створення музичних механік. Це не лише значно прискорює процес розробки, але й дозволяє зосередитися на унікальних аспектах нашої гри, не витрачаючи час на створення стандартних елементів. Мова програмування C#, яка є основним для Unity, пропонує ідеальний баланс між продуктивністю і гнучкістю. Це особливо важливо для реалізації таких складних систем, як механіка «чарівної флейти», де необхідно одночасно обробляти введення гравця, аудіо-ефекти та реакцію на такий ефект NPC.

Вибір на користь 3D, а не 2D графіки був зроблений після ретельного аналізу як технічних можливостей, так і творчих потреб проекту. Тривимірний простір надає просто неймовірні можливості для створення атмосфери - динамічне освітлення може підкреслити містичність сцен, реалістичні тіні зможуть додати глибини ігровому світу, а ефекти туману створюють потрібний настрій. Крім того, 3D дозволяє гравцям вільно досліджувати місто Гамельн і його околиці, що є ключовим для повного занурення в легенду.

2D формат, незважаючи на свою привабливість для певних типів ігор, у нашому випадку мав би ряд суттєвих обмежень. Він не дозволив би реалізувати повний потенціал використання флейти, яке є важливим елементом ігрового

процесу. Також він обмежив би наші можливості у візуальній подачі історичного антуражу, що є критично важливим для створення переконливого досвіду.

Перший підхід принципово відрізняється від існуючих рішень кількома ключовими аспектами. По-перше, ми пропонуємо справжню інтерактивність - гравець не просто спостерігає за історією, а активно впливає на її розвиток, приймаючи моральні рішення, які визначають, чи стане його персонаж рятівником чи маніпулятором. По-друге, ми використовуємо всі переваги сучасних ігрових технологій, щоб створити багатий, переконливий світ, який повністю відповідає глибині оригінальної легенди. І по-третє, наша гра заповнює очевидну прогалину на ігровому ринку, пропонуючи унікальний досвід, якого поки що ніхто не наважився створити.

Цей підхід, поєднує розуміння легенди з інноваційними ігровими механіками, дозволить мені створити не просто ще одну адаптацію, а принципово новий спосіб взаємодії з класичним сюжетом, який зацікавить як знавців оригінальної легенди, так і звичайних гравців, що шукають глибокий і незабутній ігровий досвід.

Тепер давайте поговоримо про негативні аспекти такого вибору. Однією з головних проблем є складність роботи з 3D-графікою. На відміну від 2D, тут потрібно створювати детальні моделі, продумувати анімації, текстури та ефекти, що значно збільшує час і вартість розробки. Крім того, тривимірний світ вимагає серйозної оптимізації, щоб гра стабільно працювала на різних пристроях, особливо на слабких ПК або мобільних телефонах.

Ще одним викликом є залежність від готових асетів із Unity Asset Store. Хоча вони прискорюють процес, є ризик, що гра втратить унікальність, якщо надто покладатися на стандартні рішення. Крім того, деякі асети можуть мати обмеження в ліцензіях або потребувати додаткового редагування, щоб гармонійно вписатися в ігровий світ. Технічні обмеження Unity також можуть стати на заваді. Наприклад, якщо в грі буде багато персонажів або складні сцени (як юрба щурів чи містян), це може призвести до падіння продуктивності.

Також фізика в Unity іноді поводиться непередбачувано, і доводиться витрачати додатковий час на налагодження, щоб уникнути багів. Окремо варто згадати про музичну механіку, яка є ключовою для цього проекту.

Реалізація впливу гри на флейті на оточення – технічно складна задача. Потрібно, щоб мелодії, які виконує гравець, коректно впливали на NPC, а це вимагає точної синхронізації звуку, анімацій та штучного інтелекту. Якщо зробити цю систему поверхнево, вона може виглядати непереконливо і зруйнувати враження про гру. Крім технічних аспектів, є й творчі «проблеми». Легенда про Щуролова – це містична історія з глибоким підтекстом, але в ігровій формі є ризик втратити цю атмосферу. Якщо гравець отримає надто багато свободи (наприклад, зможе просто бігати і бити всіх мечем), це може знизити напругу і драматизм сюжету. Важливо знайти баланс між відкритістю та лінійним розвитком подій, щоб зберегти дух оригінальної історії. Ще одна проблема, якщо гра не вразить гравців новаторськими механіками або глибоким сюжетом, вона може залишитися непоміченою. Щоб уникнути цих проблем, варто зосередитися на якості, а не на кількості контенту. Краще зробити менше локацій, але більш деталізованих, обдумати кожну механіку, щоб вона була цікавою, і ретельно тестувати гру на різних пристроях. Також важливо не забувати про атмосферу – музика, освітлення і загальний візуальний стиль повинні підкреслювати містичність легенди. Якщо вдасться поєднати ці елементи, гра має всі шанси стати справді незабутнім досвідом.

РОЗДІЛ 2 ПРАКТИЧНА РЕАЛІЗАЦІЯ

2.1. Опис розробленої системи/програми/алгоритму

Навіть не зважаючи на те, що я намагаюсь розробити невеликий відкритий світ всередині, в грі повинні бути певні етапи, які гравець обов'язково повинен пройти [2]. Для прикладу можна використати так моменти :

- поява в світі гри,
- знайомство з навколишнім середовищем (гравець починає подорожувати по мапі),
- перша спроба контролювати щурів (детальне знайомство з тим як і для чого можна використовувати основні механіки такого типу),
- перехід на нову локацію (скоріш за все це буде невелике місто, що повинно відтворювати, як приклад, місто Гамельн з нашої легенди),
- перше завдання,
- подорожі за межі міста (детальне дослідження інших локацій світу користувачем),
- повернення в місто,
- розроблений фінал гри (в якості відео або просто діалогу).

Тепер давайте розберемо детально кожен з прописаних пунктів. Для чого вони та способи реалізації історії за таким планом.

Коли користувач вперше з'являється в грі, він звичайно нічого не знає, максимум колись чув якусь інформацію про легенду і все, тому важливо одразу зацікавити чимось користувача, бо головна ідея нашої гри – не дійти до фіналу історії, а саме проходження гри, тобто отримувати задоволення від перебування в цьому світі та бажання використовувати спеціальні здібності головного героя. Тут треба одразу показати частину функціоналу користувачу. Наприклад, механіку переміщення, вона є базовою та обов'язковою для використання при проходженні гри.

Отже місце першої появи користувача має кардинальне значення, бо якщо, для прикладу, ми перенесемо персонажа одразу десь в центр карти, він

може піти не по сюжету гри, а кудись ще. Або одразу піде в місто, при тому, що фактично людина можливо ще не навчиться за цей час повноцінно користуватись основними механіками. Це може негативно вплинути на перше враження користувача про цей світ, заплутати та в принципі стати причиною для завершення проходження гри навіть раніше запланованого. Тому скоріш за все, я буду розміщувати головного героя десь подалі від центру карти, на окраїні, можливо в самому кутку карти. Також не зважаючи на те, що я планую розробляти саме відкритий ігровий світ, для початку історії немає нічого поганого у тимчасовому створенні певного «коридору», для обмеження користувача. Це треба, щоб користувач зміг навчитись користуватись головними механіками до того як дійде до важливого моменту або навпаки, щоб саме привести персонажа до певного сюжетного повороту.

Коли персонажа зустрічає щурів у перший раз, йому пояснюють, що саме треба робити для того, щоб почати їх контролювати (командувати та притягувати їх до себе). У щурів буде стан, при якому вони просто спочатку займаються своїми справами (бігають, стрибають, їдять щось), а потім, коли підходить щуролов та використовує свою навичку для їх контролю, перестають займатися своїми справами та слідують за ним. Проте тут треба зауважити, що для віртуального ускладнення гри, швидкість щурів менша за швидкість персонажа, тому коли головний герой буде, після початку контролю, відходити до потрібного для нього місця, а щури почнуть слідувати за ним, треба буде постійно перевіряти чи не відпав хтось з групи або йти повільніше, щоб не спугнути їх.

Перший перехід на нову локацію також має важливе значення для ігрового процесу, бо саме на цьому етапі частіше за все у користувача формується якесь своє уявлення про те, в яку саме гру він гра, чого чекати від історії цього всесвіту, чи буде цікаво користувачу в принципі витрачати свій час на проходження і так далі. В нашому випадку – це невелике місто, про яке розповідається в легенді. І хоча ми не знаємо як саме виглядало те місто в той час, але маємо власне уявлення того, що саме там може бути. В історії

описується 11-13 століття нашої ери, отже скоріш за все в цілому це буде місто оточене дерев’яними або кам’яними фортецями, з великими воротами або чимось схожим та маленькими темними будиночками всередині. Зважаючи на моє уявлення таких фортець, я уявляю, що навколо міста також є якийсь великий рів, для захисту від нападу як це часто можна побачити в фільмах, серіалах або на макетах у музеях присвячених середньовіччю (рис. 2.1 – 2.2).

Рисунок 2.1

Рисунок (2.2)

При першому завданні для користувача, йому вже треба буде використовувати механіки про які ми розповідали раніше, а також власний інтелект та увагу. Взагалі усе це буде підтримуватись якимось важливим сюжетним моментом або невеликим квестом, що не має відношення до основної історії, але є непоганим доповненням для гри в цілому.

Я поки не знаю чи у мене буде тільки одне таке завдання або декілька, але в планах також розробити якийсь сюжетний поворот, після якого користувачу треба буде тимчасово покинути місто та почати досліджувати інші локації по мапі. Для прикладу, можна привести локацію темний ліс або скелясті гори, кожна з яких буде містити власний контент, історію та ландшафт.

У мене є певне уявлення про те як саме повинен виглядати кінець цієї трагічної історії, але що саме це буде (діалог з сюжетним персонажем, записка з поясненням сюжету або короткий відеоролик вставлений в гру) я поки ще не вирішив.

Протягом усього ігрового процесу, персонажа буде супроводжувати відповідна по настрою музика, що повинна доповнювати атмосферу гри та створювати стан речей за яких користувачу буде навіть просто цікаво знаходитись у цьому світі.

2.2 Опис архітектури, функціональності та реалізації розробленого рішення

Переміщення:

Переміщення є однією з найголовніших механік для майже кожної гри зараз. В свою чергу я також її реалізував. Проаналізувавши інформацію, яку я знайшов в інтернеті, я вирішив розробити адаптивний та зручний механізм, за якого людина одразу буде розуміти, що і як робити.

Тут для переміщення використовується вже класичні кнопки (W, A, S, D):

- W – (переміщення вперед);
- A – (переміщення в ліву сторону);
- S – (переміщення назад);

- D – (переміщення в праву сторону).

Коли користувач натискає на будь-яку з цих клавіш, спрацьовує анімація руху. Окрім самої анімації, така подія супроводжується звуками кроків (ніби персонаж ходить по землі). Коли відпускає, анімація закінчується та персонаж переходить в стан спокою. Стан спокою – це стан створений для того, щоб передати персонажу властивості звичайної людини. Тобто, коли персонаж певний час стоїть на місці він робить такі речі як :

- Дихає;
- Крутиться;
- Переносить рівновагу з однієї ноги на іншу;
- Чухається;
- Виконує інші рухи.

Такі речі насправді дуже важливі, бо кожен раз заходячи гру, людина що грає ніби ставить себе на місце цього персонажу. Отже хоча стан спокою ніяк не впливає на геймплей персонажа, але створює імітацію реальності в середині гри [7, с.9][11, с. 54-56].

```
using UnityEngine;
public class PlayerMovement2 : MonoBehaviour
{
    [Header("Movement Settings")]
    public float moveSpeed = 5f;
    public float jumpForce = 7f;
    public float rotationSpeed = 10f;
    [Header("Components")]
    public Animator animator;
    public CharacterController controller;
    public Transform groundCheck;
    public Transform cameraTransform;
    [Header("Ground Check")]
    public LayerMask groundMask;
    public float groundDistance = 0.4f;
    [Header("Action Settings")]
    public float actionDuration = 2f;
    public GameObject actionObjectPrefab;
    public Vector3 objectOffset = new Vector3(-0.5f, 1f, 1f);
    [Tooltip("Rotation relative to player's forward direction")]
    public Vector3 objectRotation = new Vector3(0, 45f, 0);
    [Header("Action Object Settings")]
```

```
public Vector3 objectScale = Vector3.one;
private float gravity = -9.81f;
private Vector3 velocity;
private bool isGrounded;
private float actionTimer = 0f;
private bool isPerformingAction = false;
private GameObject spawnedObject;
private bool objectSpawned = false;
void Update()
{
   if (isPerformingAction)
   {
      actionTimer -= Time.deltaTime;
      if (!objectSpawned && actionTimer <= actionDuration * 0.5f)
      {
         SpawnActionObject();
         objectSpawned = true;
      }
      if (actionTimer <= 0f)
      {
         EndAction();
      }
      return;
   }
   UpdateGravity();
   HandleMovement();
   HandleJump();
   ApplyGravity();
}
public void StartAction()
{
   isPerformingAction = true;
   actionTimer = actionDuration;
   velocity = Vector3.zero;
   if (animator != null)
   {
      animator.SetBool("isRunning", false);
      animator.SetBool("Control_start", true);
   }
}
private void SpawnActionObject()
{
   if (actionObjectPrefab != null)
   {
      Vector3 spawnPosition = transform.position + transform.TransformDirection(objectOffset);
      Quaternion spawnRotation = transform.rotation * Quaternion.Euler(objectRotation);
      spawnedObject = Instantiate(actionObjectPrefab, spawnPosition, spawnRotation);
      spawnedObject.transform.localScale = objectScale;
      Destroy(spawnedObject, actionDuration * 0.5f);
   }
```

```
    }
    private void EndAction()
    {
        isPerformingAction = false;
        objectSpawned = false;
        if (animator != null)
        {
            animator.SetBool("Control_start", false);
        }
        if (spawnedObject != null)
        {
            Destroy(spawnedObject);
        }
    }
    private void UpdateGravity()
    {
        isGrounded = Physics.CheckSphere(groundCheck.position, groundDistance, groundMask);
        if (isGrounded && velocity.y < 0)
        {
            velocity.y = -2f;
        }
    }
    private void HandleMovement()
    {
        float moveX = Input.GetAxis("Horizontal");
        float moveZ = Input.GetAxis("Vertical");
        Vector3 cameraForward = Vector3.Scale(cameraTransform.forward, new Vector3(1, 0,
1)).normalized;
        Vector3 movement = (moveZ * cameraForward + moveX *
cameraTransform.right).normalized;
        if (movement.magnitude > 0.1f)
        {
            float targetAngle = Mathf.Atan2(movement.x, movement.z) * Mathf.Rad2Deg;
            Quaternion targetRotation = Quaternion.Euler(0, targetAngle, 0);
            transform.rotation = Quaternion.Slerp(transform.rotation, targetRotation, rotationSpeed *
Time.deltaTime);
        }
        controller.Move(movement * moveSpeed * Time.deltaTime);
        if (animator != null)
        {
            animator.SetBool("isRunning", movement.magnitude > 0.1f);
        }
    }
    private void HandleJump()
    {
        if (Input.GetButtonDown("Jump") && isGrounded)
        {
            velocity.y = Mathf.Sqrt(jumpForce * -2f * gravity);
        }
    }
```

```
    private void ApplyGravity()
    {
        velocity.y += gravity * Time.deltaTime;
        controller.Move(velocity * Time.deltaTime);
    }
}
```

Приклад реалізації в Animator Ви можете побачити нижче (рис. 2.3).

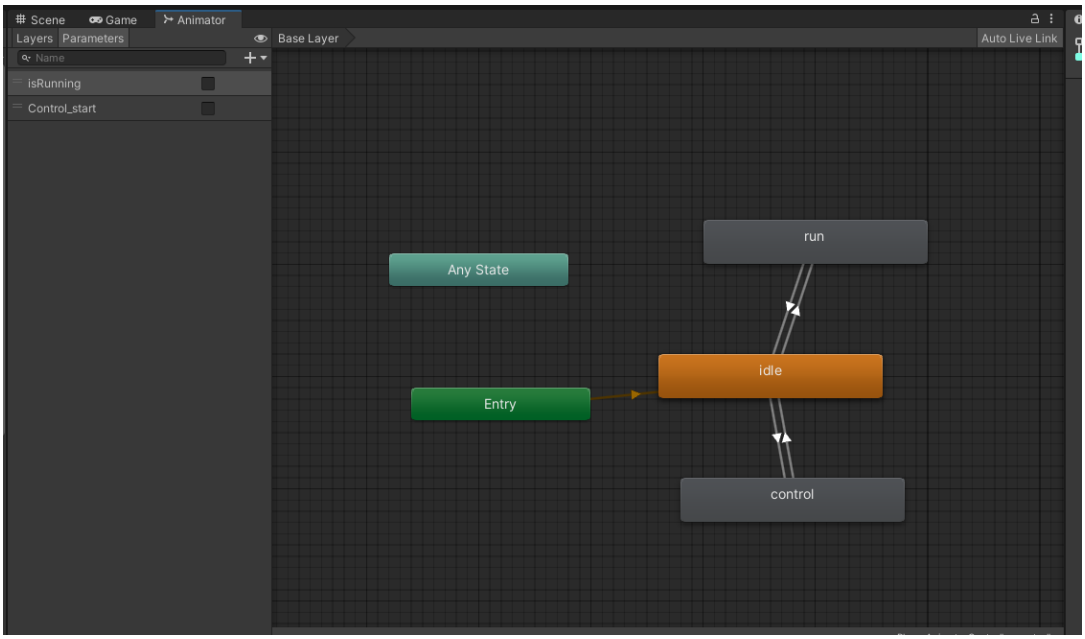


Рисунок 2.3

Насправді я не створював анімації самостійно, а взяв їх з інтернету. Але напряму перенести анімації у мене не виходило, тому я почав використовувати програму Blender. Туди я переніс моє тіло персонажа, після чого створив поверх нього спеціальний скелет. Він як і людське тіло складається з кісток, кожна з яких відповідає за певну частину. Коли вони нанесені по всьому тілу, ними можна маніпулювати як реальними та створювати реалістичні анімації. Але зважаючи на те, що ми переносимо чужі анімації, а не створюємо свої, то просто треба прикріпити їх, до тіла (кісток), іншу частину програма зробить самостійно. Після цього ми прокручуємо анімацію та перевіряємо чи все працює нормально (рис. 2.4).

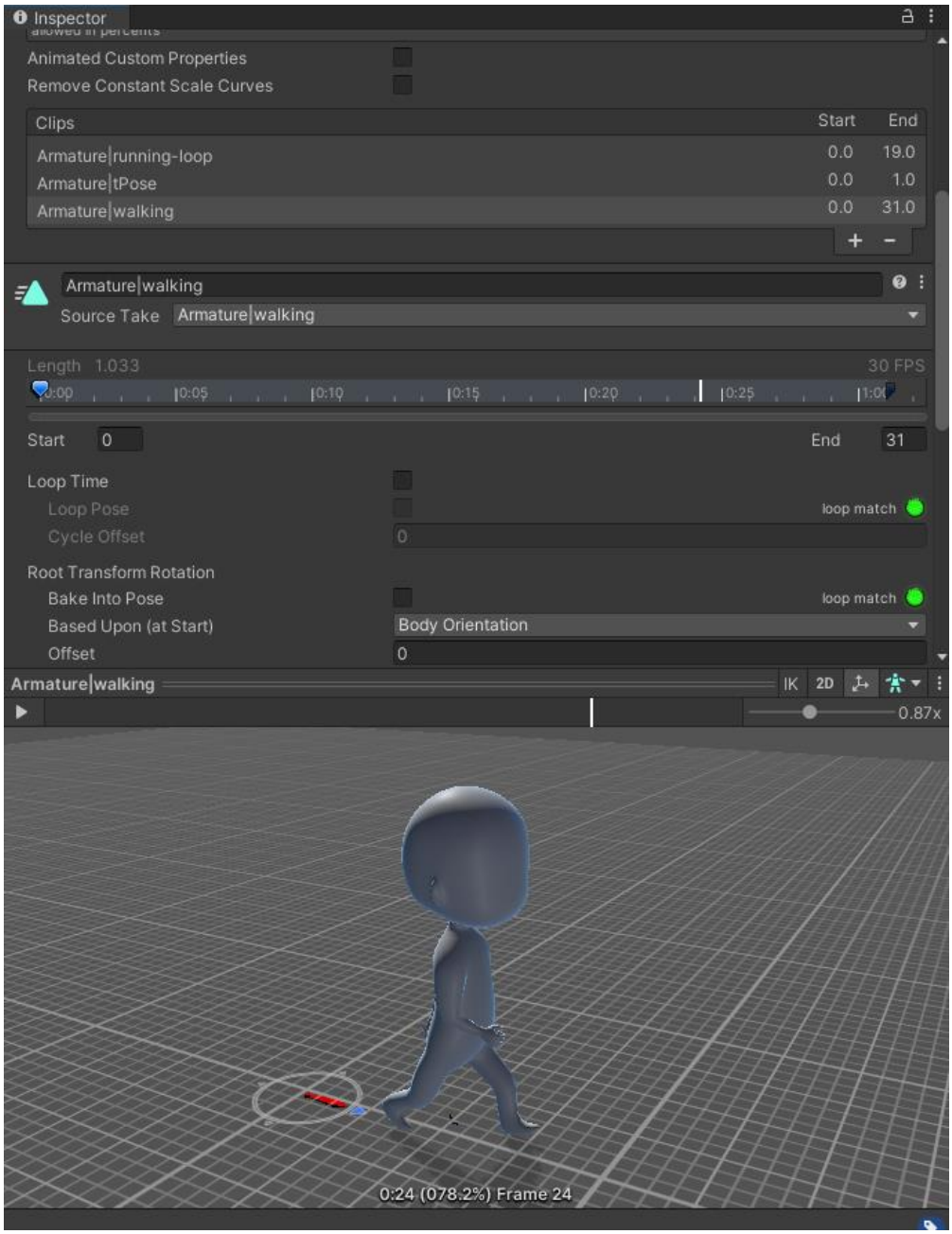


Рисунок 2.4

Коли я робив анімацію руху, то просто переніс анімацію як є, що не можна зробити з анімацією стану спокою, бо людина може знаходитись в грі годинами та не чіпати жодної з кнопок, а анімація займає від 5 до 30 секунд. Отже наш наступний шаг доволі очевидний, прописати відповідну умову та зробити анімацію зацикленою, це дозволить нам спостерігати за змінами на екрані навіть після багатьох годин без участі людини.

Також у нас є анімаціє стрибка при натисканні на SPACE. В такі моменти інші анімації зупиняються та починає працювати саме анімація стрибка.

Контроль щурів:

При натисканні на клавішу E, я створив можливість контролювати щурів. Як це працює? У нас є навколо персонажа невидиме поле (зона). Коли об’єкт з тегом «Rat», знаходиться в цій зоні та людина натискає на відповідну клавішу, цей об’єкт починає наближатись до персонажа (рис. 2.5).

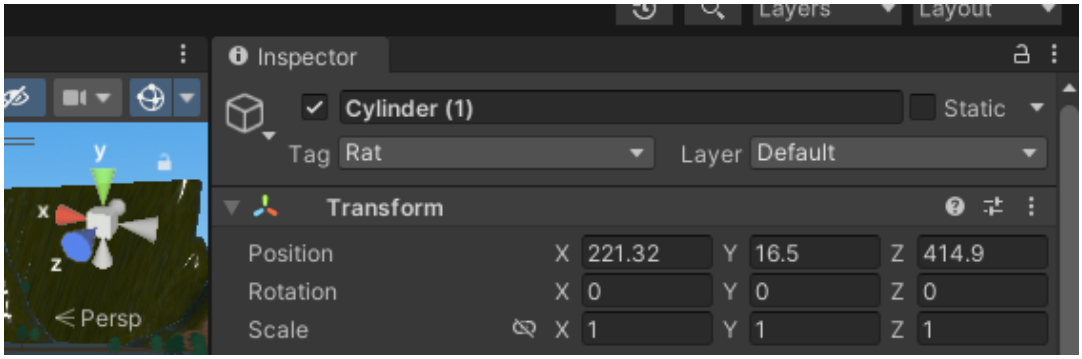


Рисунок 2.5

Нижче наведено, як саме виглядає скрипт.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using Cinemachine;
public class FollowPlayer : MonoBehaviour
{
    [Header("Налаштування")]
    public float followRadius = 5f;
    public float stopFollowingRadius = 15f;
    public float stopDistance = 1f;
    public string targetTag = "Rat";
    public float followSpeed = 2f;

    [Header("UI Елементи")]
    public Text followerCountText;
    public RawImage background;
    private List<RatFollower> followers = new List<RatFollower>();
    void Update()
    {
        if (Input.GetKeyDown(KeyCode.E))
        {
            FindNewFollowers();
        }
        MoveFollowers();
        UpdateUI();
    }
```

```
void FindNewFollowers()
{
   Collider[] hits = Physics.OverlapSphere(transform.position, followRadius);
   foreach (Collider hit in hits)
   {
      if (hit.CompareTag(targetTag))
      {
         RatFollower ratFollower = hit.GetComponent<RatFollower>();
         if (ratFollower != null && !followers.Contains(ratFollower))
         {
            ratFollower.isControlled = true;
            ratFollower.target = transform;
            ratFollower.followSpeed = followSpeed;
            followers.Add(ratFollower);
         }
      }
   }
}
void MoveFollowers()
{
   for (int i = followers.Count - 1; i >= 0; i--)
   {
      RatFollower follower = followers[i];

      if (follower == null || follower.gameObject == null)
      {
         followers.RemoveAt(i);
         continue;
      }
      float distance = Vector3.Distance(transform.position, follower.transform.position);
      if (distance > stopFollowingRadius)
      {
         follower.isControlled = false;
         followers.RemoveAt(i);
      }
   }
}
void UpdateUI()
{
   followerCountText.text = followers.Count.ToString();
   bool shouldShow = followers.Count > 0;
   followerCountText.gameObject.SetActive(shouldShow);
   background.gameObject.SetActive(shouldShow);
}
void OnDrawGizmosSelected()
{
   Gizmos.color = Color.green;
   Gizmos.DrawWireSphere(transform.position, followRadius);
   Gizmos.color = Color.red;
   Gizmos.DrawWireSphere(transform.position, stopFollowingRadius);
```

```
    }
}
```

В той самий час, в лівій нижній частині екрану з’являється лічильник кількості контрольованих щурів на даний момент часу, а також спрацьовує анімація персонажа, коли він підносить флейту до себе та починає грати на ній. Тепер коли персонаж рухається, усі об’єкти будуть слідувати за ним. Але є ще друге поле навколо персонажа, це поле потрібне для відслідковування щурів під нашим контролем. Коли щур покидає його, з нього спадає ефект контролю та він починає рухатись вже окремо від усіх. Таким чином персонаж повинен постійно контролювати те, на скільки швидко він ходить або бігає (рис. 2.6 - 2.7).

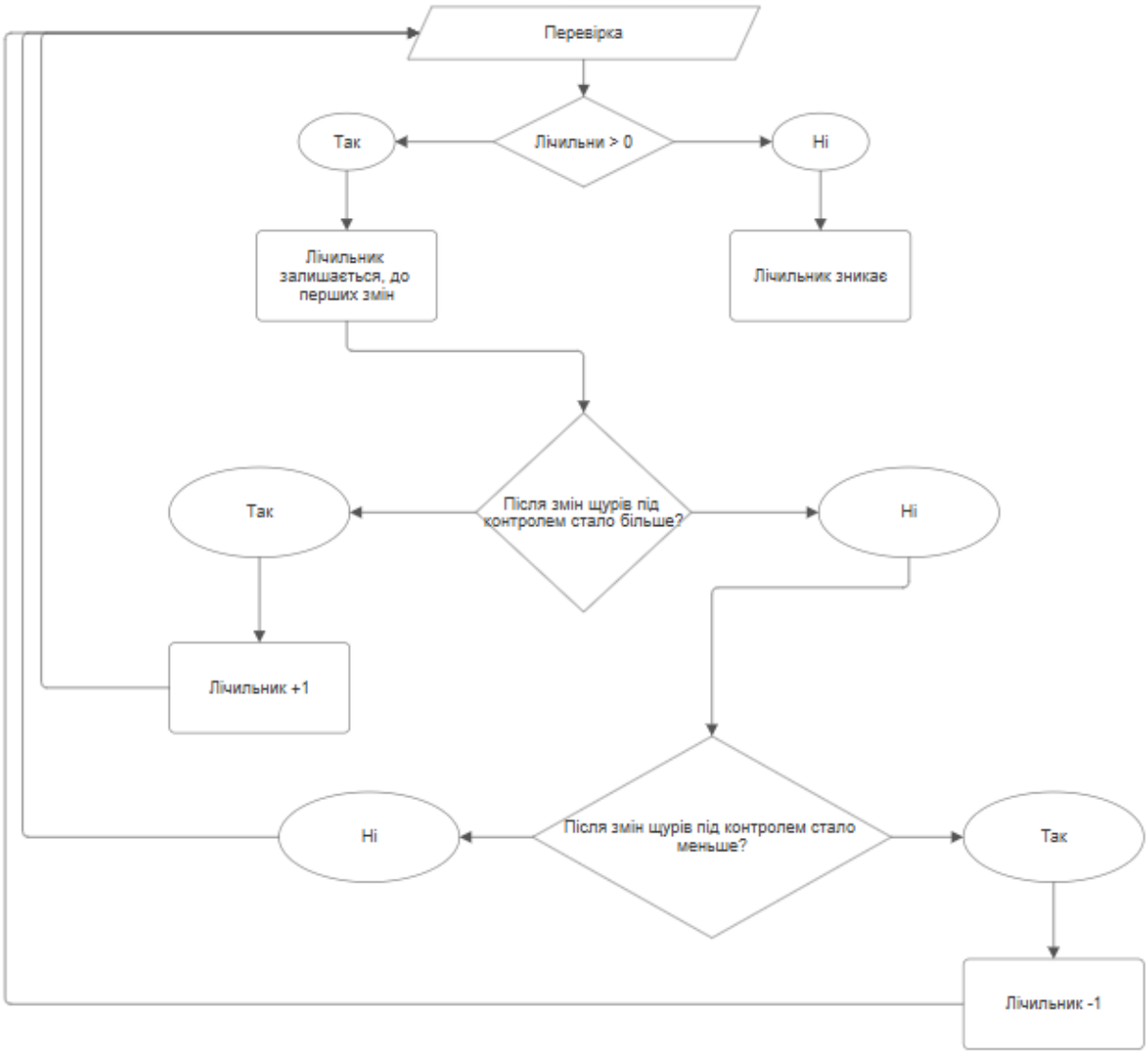


Рисунок 2.6

Але справа в тому, що наші щури, для Unity – це просто ще один об’єкт. Тому, крім того, що дозволяти відслідковувати персонажа. Нам також треба рахувати, яке відношення та в якому напрямку має певний щур. Для цього я створив ще один скрипт, який треба присвоїти вже самому щуру, а не персонажу (додаток А).

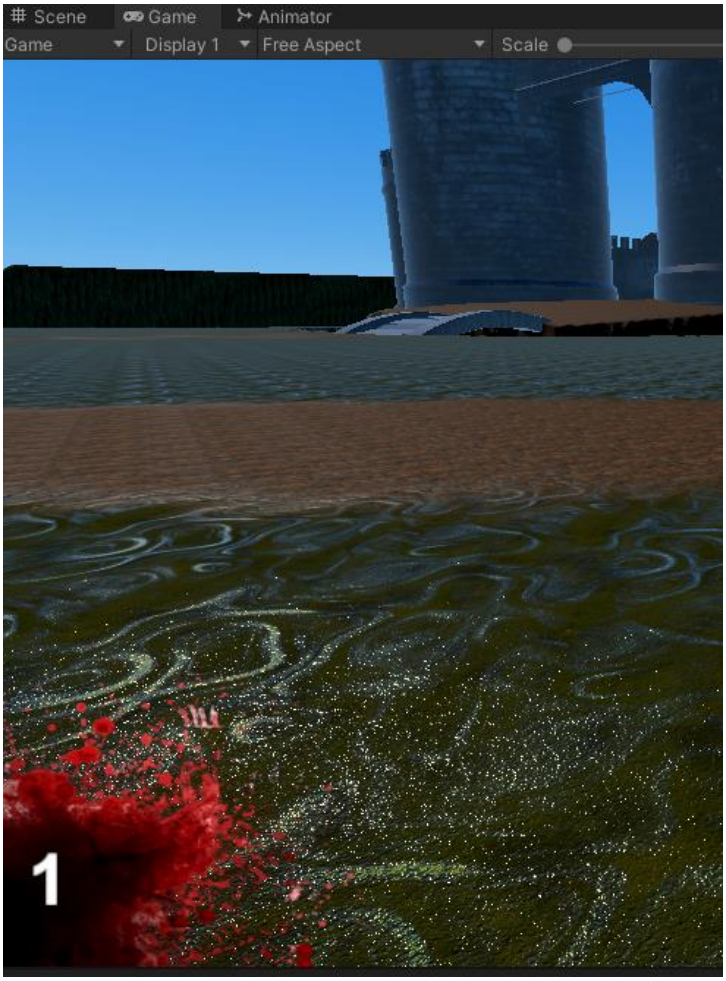


Рисунок 2.7

Це і є головним механізмом моєї гри. А загальний механізм змушує гравця дуже обачно використовувати цю навичку.

Розробка ландшафту:

Для розробки ландшафту я використовував спеціальний інструмент Terrain, його особливість в тому, що ти можеш за дуже короткий час з однієї плоскої поверхні створити справжню копію реального ландшафту з перепадами висот, кольором поверхонь, а також прямо на цю поверхню можна додавати дерева, траву та інші елементи, що дуже нагадують живу природу [5, с.7].

Розглянемо особливості створення кожного типу, а також, що саме повинно бути в моєму проекті з цього та в якій формі.

Перепади висот, тобто висота землі, що дозволяє створювати відкритий широкий простір або справжні гори. Як я вже казав раніше, в моєму проекті навіть якщо і повинен бути відкритий світ, але треба розуміти, що першій 5 хвилин геймплею повинні бути «коридорними» (лінійними), щоб користувач не міг піти подорожувати картою раніше, а ніж я поясню йому як усе тут працює. Тому на початку треба зробити стіни, в нашому випадку гори, навколо початкової точки появи користувача.

Також крім початку, я в принципі планую створити стилізовані гори в іншій частині карти. Це буде додавати карті вишуканості та різновиду, фактично зробивши як окрему локацію для дослідження (рис. 2.8).

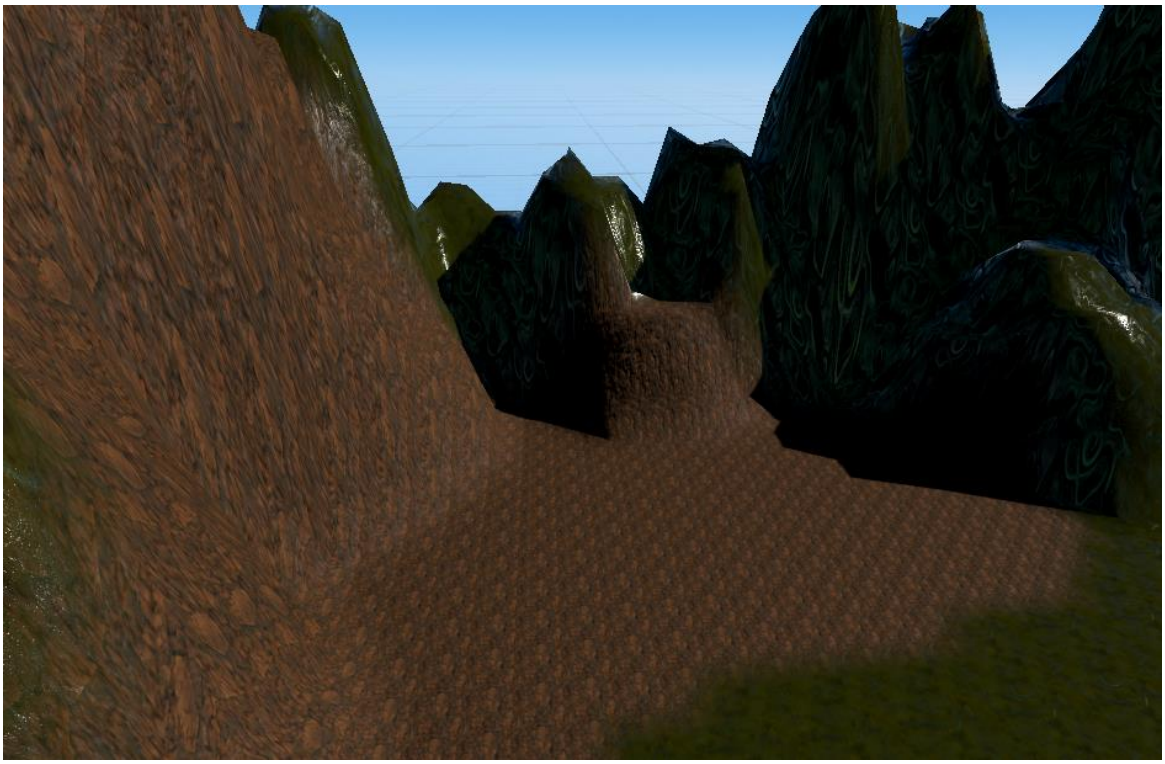


Рисунок 2.8

Уявимо, що ми вже зробили усі потрібні або заплановані перепади висот. Отже наступний крок, якось відділити їх від сірої гамми. В нагоді нам стане можливість фарбувати у різні кольори певні окремі частини локацій. Проте ми плануємо не просто відділити, а й зробити максимально реалістичну мапу, тому замість звичайних кольорів ми будемо використовувати різні текстури, що

нагадують траву, скелі, сніг і т. д. Що і буде формувати скелясту місцевість нашого ландшафту (рис. 2.9).

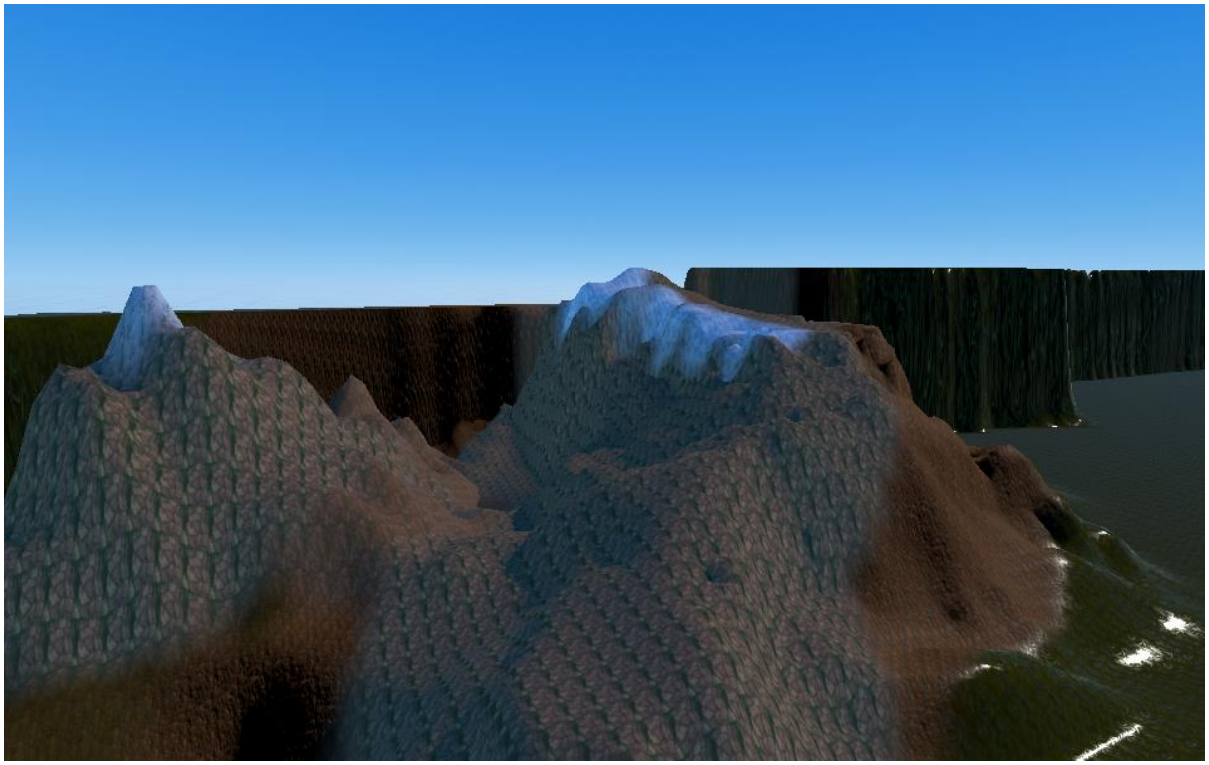


Рисунок 2.9

Наступний пункт – зелень. Тобто трава, кущі та дерева. В налаштуваннях Terrain для Unity є окремі вкладки для трави та дерев. Вони допомагають просто завантажити в окреме поле нашу 3D модель та використовувати її вже не по-одинці, а висаджувати цілими групами (рис. 2.10).

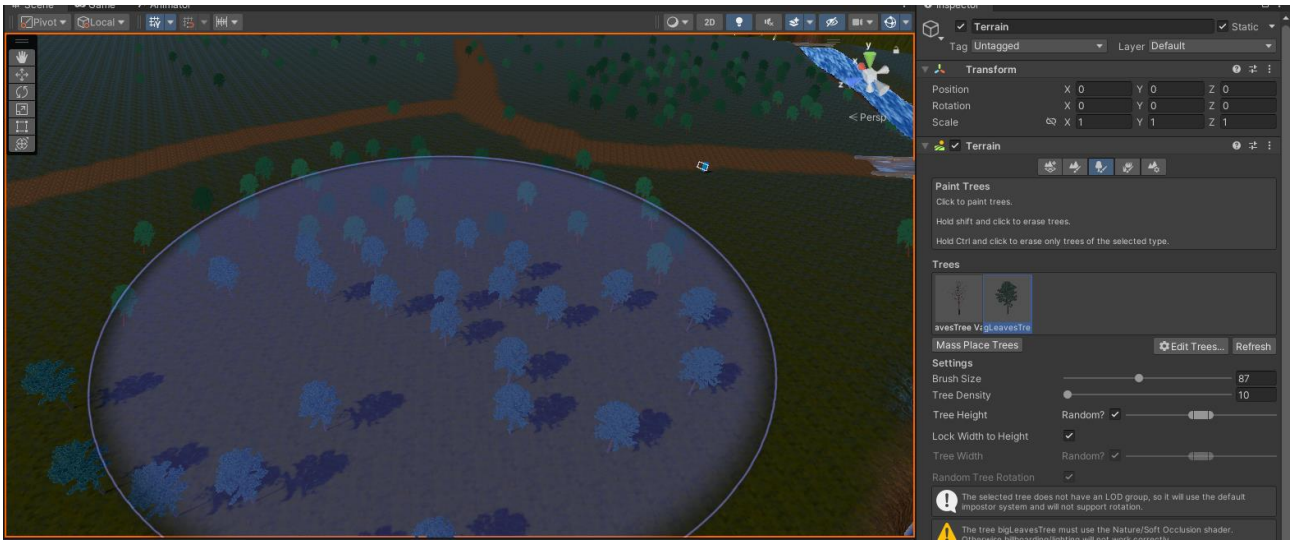


Рисунок 2.10

Формування міста:

Історія нашого всесвіту відбувається у період раннього середньовіччя, тому і місто повинно мати відповідний характер. Фортеці, кам'яні пошарпані будівлі, широкий та високий прохід у місто та глибокий рів від нападників навколо. Спочатку я спробував зайнятись створенням будівель та фортець, але у мене погано виходило і перейшов на пошук моделей з інтернету. Як я вже казав на початку, однією з справді важливих переваг Unity, можна назвати наявність Unity Asset Store – власний сайт з великою кількістю якісних безкоштовних моделей, що можна завантажити напряму в Unity, якщо підключити свій акаунт, що Ви використовуєте в Unity.

Тут я знайшов необхідну для мене асет з файлами моделей частин замку. Після чого я додав її в мою власну бібліотеку та завантажив до проекту прямо через сайт (рис. 2.11).

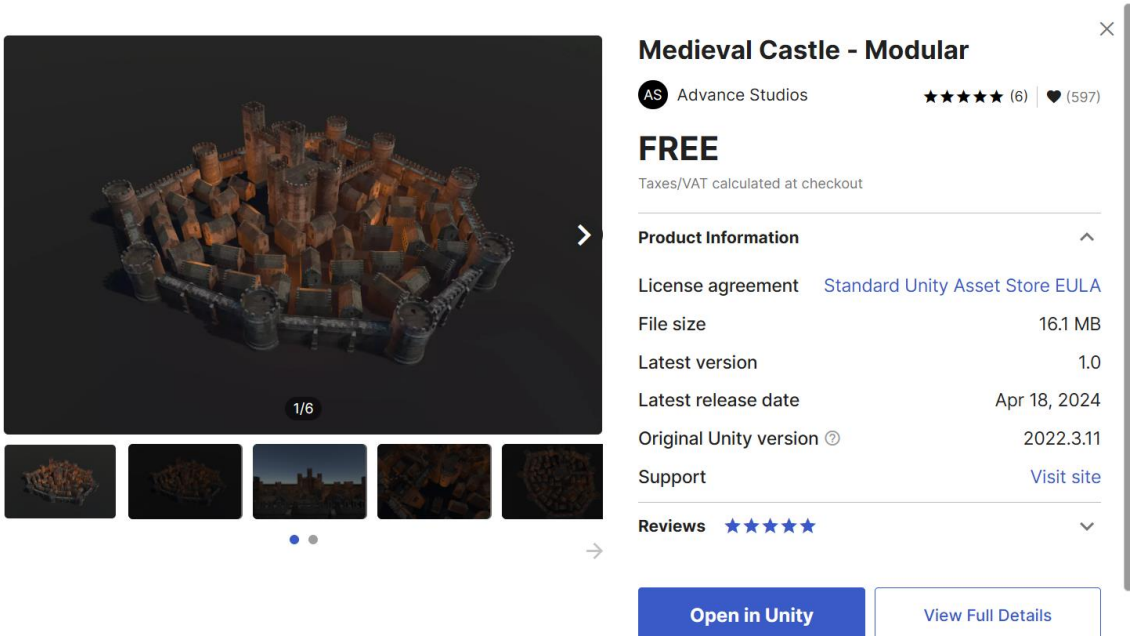


Рисунок 2.11

Створення води:

Одним з найскладніших етапів моєї розробки, стало створення реалістичної води в Unity.

По-перше, за основу реалістичної води я також взяв інший асет з бібліотеки Unity Asset Store. Але при завантаженні стикнувся з неочікуваною проблемою. Скачані файли відображались не вірно, тобто сам Unity не міг обробити дані, що лежали в цій бібліотеці. Тому я пішов читати документацію.

В результаті довгого та ретельного пошуку, я зрозумів я к виправити цю помилку. Виявилось, що деякі файли можуть містити власні стилі, що за якихось причин, заборонені стандартами системи, тому це навіть не помилка, а застереження щодо файлів. Щоб я все таки мав доступ до цих файлів я скачав спеціальну системну бібліотеку, що дозволяє підключати спеціальні елементи в моделі (рис. 2.12) [13].

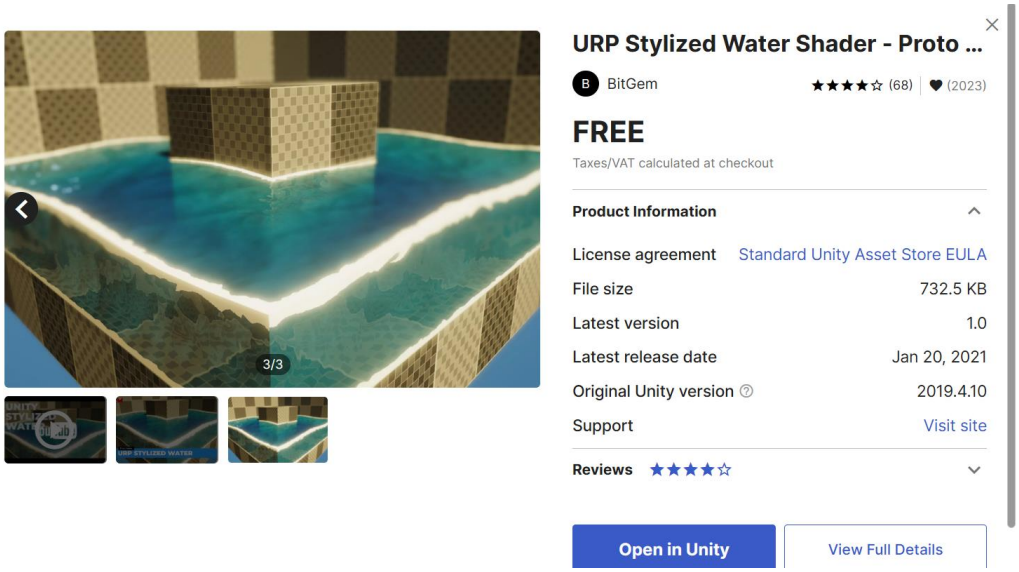


Рисунок 2.12

Людина яка створила цю бібліотеку залишила відео з поясненням того, як саме підключити воду. Зробивши усе по відео, я отримав те, що хотів та додатково розібрався в керуванні системними файлами проєкту. Під кінець, я використовуючи можливість редагування створив довгу глибоку яму, яку накрив на певному рівні штучно зробленою водою (рис. 2.13).

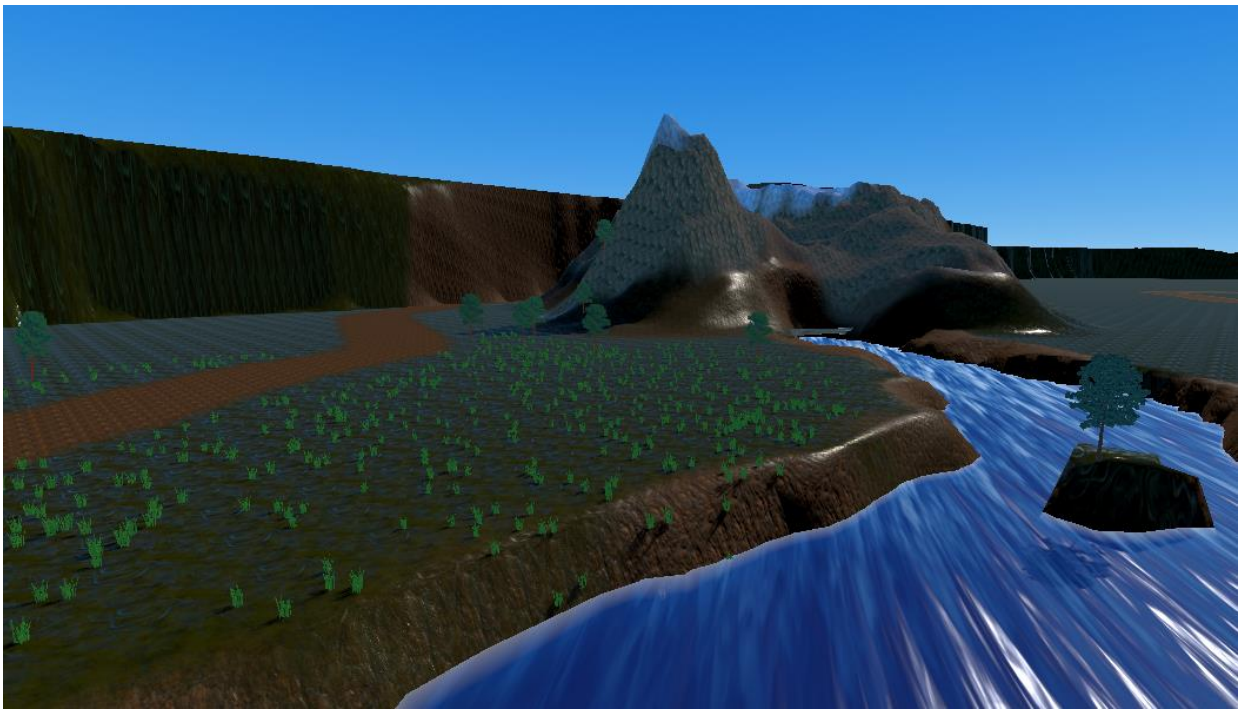


Рисунок 2.13

Меню гри:

Я довго думав, яким чином мені створити головне меню гри, бо це фактично «лице» гри. Спочатку хотів скопіювати ідею інших розробників, яку бачив в інтернеті, а саме відображення на задньому фоні локації гри створивши меню прямо на тій самій сцені, пізніше просто скривати її для користувача та знову показувати коли він захоче вийти в меню [4, с.5]. Проте пізніше змінив свою думку, UI мого головного меню я розробив завдяки створенню квадратної фотографії через ШІ. Для цього я знайшов відповідний сайт в інтернеті та прописав prompt для подальшої генерації. Після отримання фото в якості результату, вирішив її графічно вдосконалити, через програму Canva.

Фото було створено у графічному стилі з жовтим та чорним кольорами, що буде підкреслювати та наштовхувати на ідеї про своє бачення переказу якоїсь древньої історії. Такий стиль я часто зустрічав у різних старих книжках. Таких як «Діти капітана Гранта», «15-річний капітан», «Дон Кіхот» та «Алхімік» (рис.2.14).

Саме меню я зробив легким для розуміння (без великої кількості не потрібних кнопок) і адаптивним під різні розміри екрану, не залежачи від того, який у Вас ПК, завдяки чому кнопки не будуть з’їжджати [14][15].

Рисунок 2.14

Для створення сторінки налаштувань, я створив два типи меню. Перший-основний, що буде показуватись одразу на екрані. Інший – налаштування, на цю сторінку можна буде перейти з головної. При переході, основне меню закривається, а на його місці з’являється інше. Для налаштувань я використав три параметри: чутливість (для миші), музика (для регулювання гучності пісень та музики) та звук (для інших ефектів пов’язаних зі звуком) (рис. 2.15).

При поверненні та переході на інші сцени, визначені параметри як правило одразу зберігаються та використовуються.

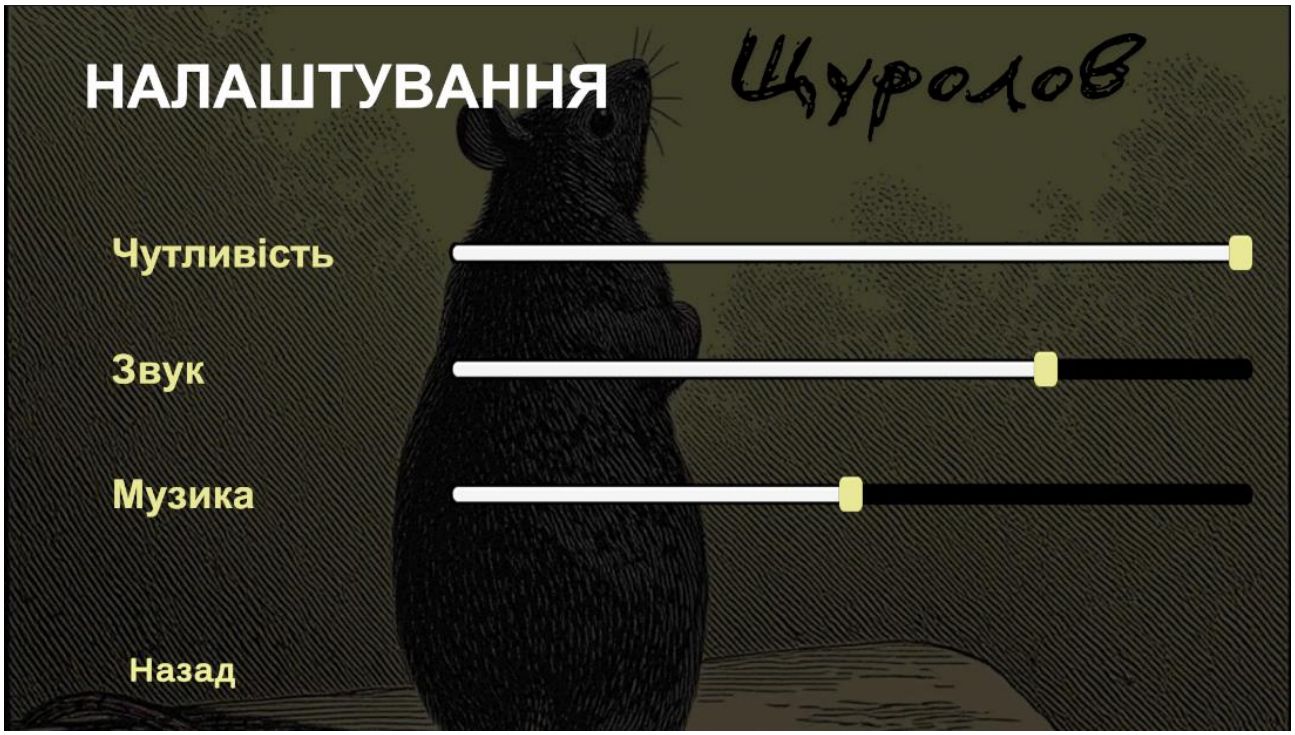


Рисунок 2.15

Музика та звуки в грі:

Музика є важливою частиною ігрового процесу. Вона впливає на загальне сприйняття, доповнює атмосферу та дає можливість глибше передати емоції гри або її історію. В поганому варіанті, вона може навпаки, відштовхувати від себе.

При додаванні в проект музики, я вирішив притримуватись декількох важливих факторів. По-перше, музика повинна грати на задньому фоні постійно, але бути не занадто гучною для сприйняття. Це не головне при активні грі, а скоріше один із головних додаткових аспектів, що впливає на настрій людини, що вирішила пограти. По-друге, пісень повинно бути декілька, бо зациклена одна пісня може, спочатку гарно вплинути на користувача, а пізніше, набридне, та почне навпаки дратувати.

Але я не вмію створювати музику, тому знов таки почав шукати ідеї в інтернеті, де наткнувся на ШІ, який може мені допомогти в цьому – Suno.

Як і в ситуації з генерацією фотографій, тут мені потрібно було написати prompt. Але в даному випадку, стосовно музики. Наприклад, якого типу я хочу, можливо, які інструменти я б хотів використовувати. Мій вибір зупинився на

флейті та піаніно в тематиці Dark Fantasy, щоб передати похмуру атмосферу навколо. Ну і звісно флейта, через те, що це саме той інструмент, яким по історії користувався щуролов (рис. 2.16).

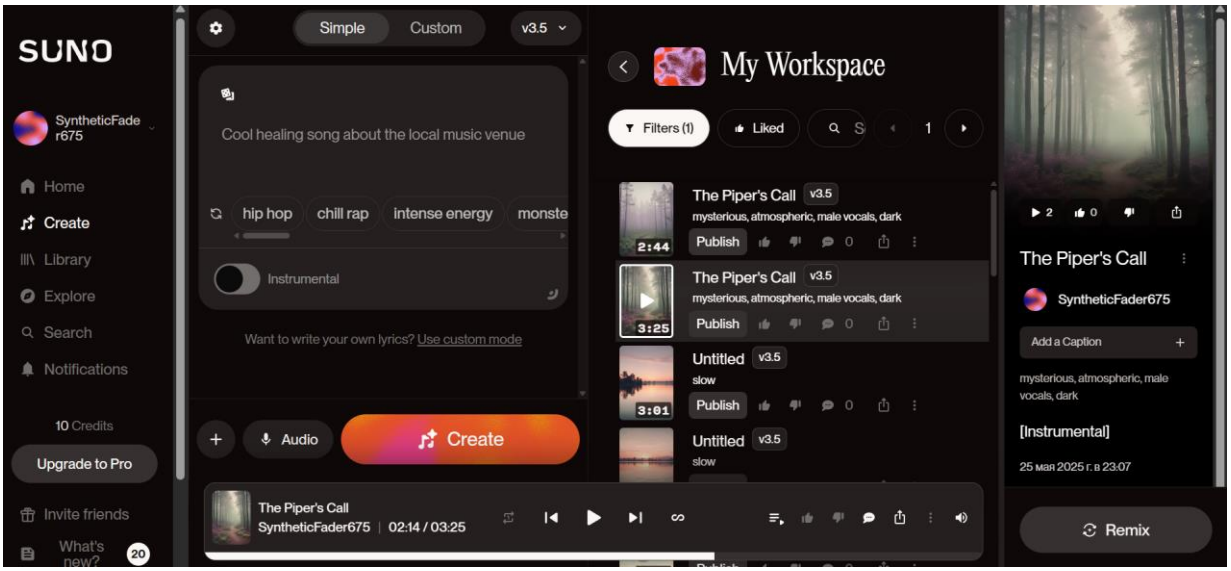


Рисунок 2.16

А щодо інших звуків: звуку бігу або звуку флейти під час початку контролю щурів. Їх я просто шукав в інтернеті. Додатково в процесі пошуку, дізнався, що деякі програмісти, розробники, дизайнери та просто фанати ігор створюють власні повноцінні асети з великою кількістю музики під ігри в різних стилях.

Діалогова система:

Тепер коли користувач підходить до певних NPC – відбувається скрипт, після виконання якого, у користувача забирають камеру та примусово змінюють її положення. Це зроблено для того, щоб користувач міг повністю зосередитись на діалозі з NPC.

Під час діалогу користувач зможе дізнатись про певний сюжет гри, а також отримати завдання. Я зробив так, що воно з’явиться одразу після того, як персонаж закінчить діалог. Також треба зауважити, що після завершення розмови, камера буде повернена користувачу для подальшого користування.

Спочатку я збирався створити повноцінні вітки діалогів для персонажа, але після кількох не дуже вдалих спроб, я відмовився від такої ідеї.

РОЗДІЛ 3 ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТИ

3.1. Результати експериментів і тестування

Вирішення проблем з FPS:

Після того, як я почав створювати повноцінні локації для моєї гри, я зіткнувся з новими викликами. Переходячи в ігровий режим (режим звичайного користувача), на деяких локаціях у мене з'являлись суттєві проблеми з падінням FPS.

Причину таких проблем, можна зрозуміти одразу. Для цього треба згадати, що таке FPS та від чого він залежить. Це кількість кадрів, які гра відображає за одну секунду. Цей показник є основним індикатором продуктивності гри та безпосередньо впливає на ігровий досвід:

- Нижче 30 FPS – гра стає зламаною, рухи виглядають не дуже якісними, що сильно псує враження.
- 30–60 FPS – прийнятний рівень для казуальних ігор, але для шутерів або гоночних симуляторів цього може бути недостатньо.
- 60 FPS і вище – ідеальний варіант, забезпечує максимальну плавність.
- 144 FPS і більше – стандарт для кіберспортивних ігор, де кожен кадр впливає на реакцію гравця.

У моєму проекті спочатку спостерігалися проблеми з продуктивністю – FPS періодично падав до 25–30, що робило гру менш комфортною. Отже, для вирішення проблеми, треба просто видалити те, що нам заважає та навантажує систему. Важливо зауважити, що ми кількість кадрів значно просідає тільки в той момент, коли в полі зору користувача знаходяться об'єкти, що навантажують систему. В нашому випадку – трава.

Як я вже показував раніше, Terrain в Unity дозволяє розміщати велику кількість однакових об'єктів у вибраній зоні. І я звісно використав це собі на користь, розмістивши велику кількість зелені по всій мапі. Що і спричинило для мене проблеми такого типу.

Після детального пошуку варіантів для вирішення цього питання, я все таки знайшов декілька, проте кожен з них потребує значну кількість часу на виконання та не є гарантом повного вирішення таких проблем [6]. Я ж в свою чергу, вирішив не витрачати на таке час та спробував замінити мої вибрані об’єкти трави, на інші, що трохи менше навантажують систему, але значного росту FPS у так і не з’явилось. В результаті, у мене не було іншого вибору, окрім як повністю відмовитись від трави, залишивши при цьому лише дерева, які хоч і більші за розміром, але стоять не в такій великій кількості по мапі.

Інший раз я стикнувся з такою проблемою, коли почав створювати воду. Взагалі її в принципі складно зробити реалістичною, а на слабких ПК або ноутбуках, ще складніше.

Як перший варіант, я знайшов динамічну воду через Unity Asset Store, що справді нагадувало за своєю структурою реальну. Додав її до своєї бібліотеки та завантажив файли в проєкт. І хоча це зайняло багато часу, я все таки знайшов як реалізувати її в проекті. Коли усе було зроблено, запустив проект для перевірки, а отримав повідомлення про помилку завантаження та повноцінний виліт з Unity (рис. 3.1).

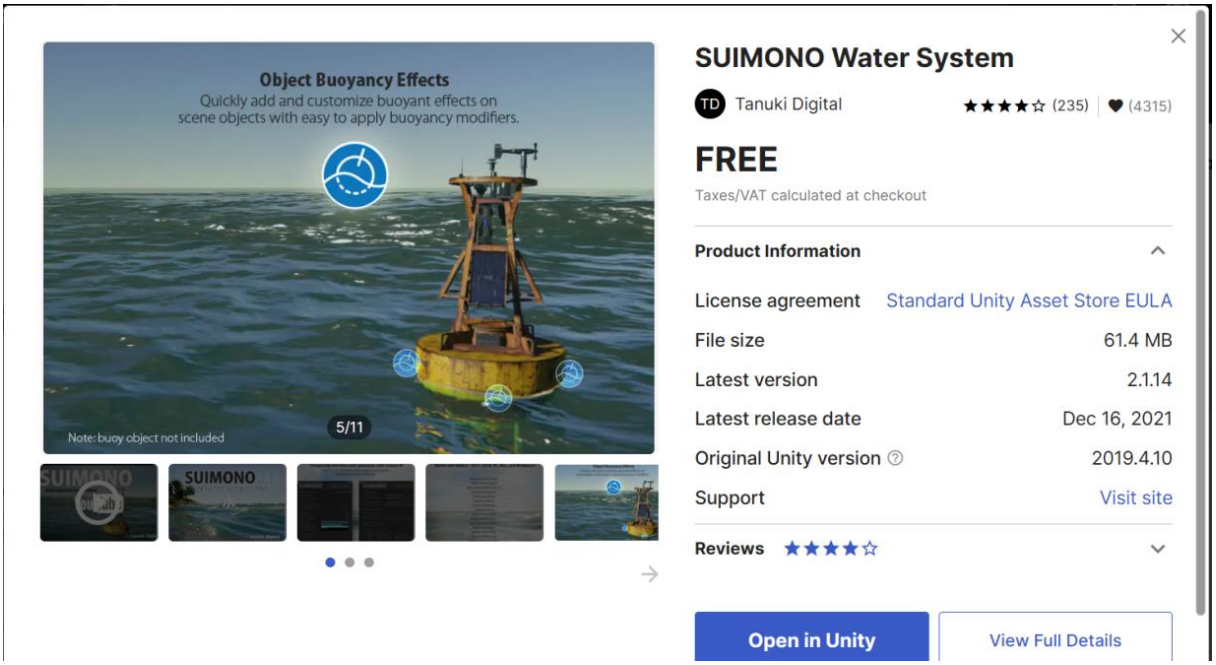


Рисунок 3.1

Зрозумівши, що є не можу собі дозволити додати такий елемент в свою гру, я одразу прийняв рішення щодо видалення його з проєкту. Та пішов шукати інші варіанти. Зупинився на симуляції води, яка не була динамічною, проте мала власну анімацію, що нагадували коливання хвиль, яка створювала ефект схожості. І хоча спочатку такий варіант мені не сподобався, вибору я не мав. Вже цей спосіб у мене спрацював нормально. Навантаження все ще було, але не на стільки критичне як раніше. Тут зміни вже працювали не динамічно, а за певним скриптом, що вказувало, яку висоту може мати хвиля, як саме буде переміщуватись цей ефект на яку відстань воно спрацьовує.

```
#region Using statementsusing System.Collections;
using System.Collections.Generic;
using UnityEngine;
#endregion
namespace Bitgem.VFX.StylisedWater
{
   [AddComponentMenu("Bitgem/Water  Volume (Transforms)")]
   public class WaterVolumeTransforms : WaterVolumeBase
   {
      #region MonoBehaviour events
      private void OnDrawGizmos()
      {
         if (!ShowDebug)
         {
            return;
         }
         // iterate the chldren
         for (var i = 0; i < transform.childCount; i++)
         {
            // grab the local position/scale
            var pos = transform.GetChild(i).localPosition;
            var sca = transform.GetChild(i).localScale / TileSize;
            // fix to the grid
            var x = Mathf.RoundToInt(pos.x / TileSize);
```

```
            var y = Mathf.RoundToInt(pos.y / TileSize);
            var z = Mathf.RoundToInt(pos.z / TileSize);
            var drawPos = new Vector3(x, y, z) * TileSize;
            var drawSca = new Vector3(Mathf.RoundToInt(sca.x), Mathf.RoundToInt(sca.y),
Mathf.RoundToInt(sca.z)) * TileSize;
            drawPos += drawSca / 2f;
            drawPos += transform.position;
            drawPos -= new Vector3(TileSize, TileSize, TileSize);
            // render as wired volumes
            Gizmos.DrawWireCube(drawPos, drawSca);
        }
    }
    private void OnTransformChildrenChanged()
    {
        Rebuild();
    }
    #endregion
    #region Public methods
    protected override void GenerateTiles(ref bool[,,] _tiles)
    {
        // iterate the chldren
        for (var i = 0; i < transform.childCount; i++)
        {
            // grab the local position/scale
            var pos = transform.GetChild(i).localPosition;
            var sca = transform.GetChild(i).localScale / TileSize;
            // fix to the grid
            var x = Mathf.RoundToInt(pos.x / TileSize);
            var y = Mathf.RoundToInt(pos.y / TileSize);
            var z = Mathf.RoundToInt(pos.z / TileSize);
            // iterate the size of the transform
            for (var ix = x; ix < x + Mathf.RoundToInt(sca.x); ix++)
            {
                for (var iy = y; iy < y + Mathf.RoundToInt(sca.y); iy++)
                {
```

```
                for (var iz = z; iz < z + Mathf.RoundToInt(sca.z); iz++)
                {
                    // validate
                    if (ix < 0 || ix >= MAX_TILES_X || iy < 0 | iy >= MAX_TILES_Y || iz < 0 || iz >=
MAX_TILES_Z)
                    {
                        continue;
                    }
                    // add the tile
                    _tiles[ix, iy, iz] = true;
                }
            }
        }
    }       #endregion
  }
}
```

Правильне переміщення камери:

Після того як я реалізував переміщення персонажа по карті, я фінальне тестування цієї механіки, під час якого, дізнався про, доволі очевидну на перший погляд, проблему – повороти камери. Тобто мера завжди дивилась на однакову відстань з кожної сторони. Це ніяк не впливає на ігровий процес, але якщо максимально відтягнути сенсор миші вниз, гравець бачить усе під картою, тобто усі локації, що може також не сподобатись користувачу.

Я згадав, що вже бачив ефект в іграх по типу Genshin Impact, коли камера наближається або віддаляється від користувача, залежно від точки зору, при цьому не потребуючи використання додаткових клавіш. В ютубі я зміг знайти приклад реалізації, що був максимально схожий на той, що я бачив раніше та зробив його за прикладом (рис. 3.2).

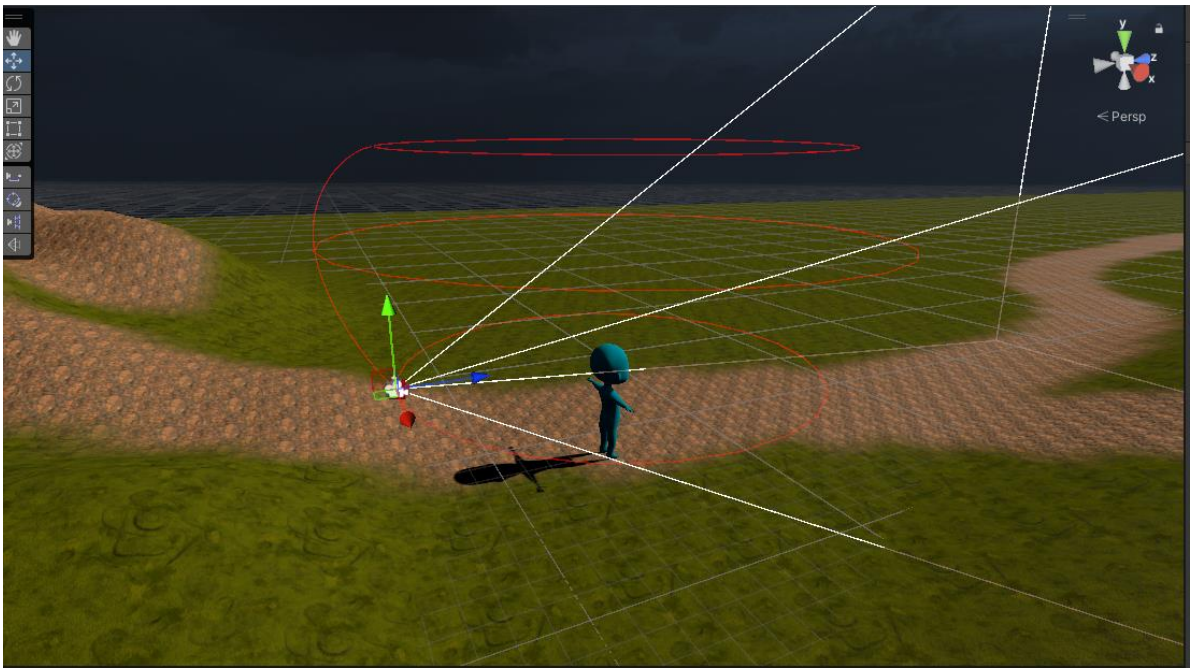


Рисунок 3.2

Для цього, замість звичайної камери я створив зону, спеціально створену для камери, в якій вказав, що вона завжди повинна спостерігати за гравцем та одночасно переміщатись тільки по краєм зони. Таким чином, фокус камери завжди на користувачі та паралельно ми можемо не хвилюватись про те, як користувач буде крутити камерою [8, с.6].

Додавання власного ШІ:

Однією з найшвидших за розвитком спеціальностей в світі – ШІ. Ще 5 років назад, ніхто не думав, що ми тепер будемо використовувати їх так часто для повсякденних справ. Тому звичайно у мене з’явилось бажання створити власний, навіть якщо він не буде дуже прогресивним або виконувати насправді важливі справи. Просто для практики, щоб перевірити на скільки легко це зробити.

Я передивився велику кількість відео в інтернеті та зупинився на ідеї про ШІ з системою балів [1][5, с.5]. Це система, основа якої базується на виділенні задач та балів при виконанні задач поставлених творцем. А також створенню певних обмежень, правил та діапазону функціоналу. Наприклад, ми даємо ШІ можливість переміщатись по території та ставимо йому задачу, перейти по мосту, що знаходиться далі, якщо пройти не в правильному місці – впадеш за

межі нашої карти. Тепер створимо обмеження та стимул (бали). Якщо він не наближається в сторону моста, у нього забирають певну кількість балів кожні 10 секунд. Якщо падає за карту – віднімається велика кількість балів. Якщо правильно перейшов, то отримує велику кількість балів.

Завдяки такій системі, через певний час (8-12 годин) він навчиться робити потрібні тобі речі. Можливо не ідеально, але прогрес точно буде. За таким прикладом і створюються ШІ в Unity.

3.2 Порівняння з існуючими рішеннями

Давайте повернемося до FPS. Як правильно оптимізувати ігри, щоб при додаванні нових ресурсів, система не доводила до зламаності. Давайте розглянемо декілька варіантів, з умовою про те, що ми не можемо оновлювати наш ПК або ноутбук на новіший, з кращою внутрішньою системою:

1. Розробка трохи меншого, а ніж було заплановано по розміру, відкритого світу. Створення коридорів, завдяки яким нам потрібно буде вирішувати проблеми з оптимізацією тільки на обмеженій території.
2. Якщо ми все таки розробляємо масивний відкритий світ, можна зробити невидиму зону навколо гравця. Все що в неї входить, повинно бути показано, а що ні – сховано, окрім певних елементів, що не можна ховати. Для прикладу основу локацій. Таким чином ми витрачаємо ресурси тільки на обмежену територію навколо. В результаті чого, для користувача усе буде виглядати нормально. Навіть якщо це не так, але при цьому, кількість FPS не буде сильно падати. Прикладом такої оптимізації можна назвати нашу українську гру S.T.A.L.K.E.R. 2:Heart of Chernobyl (рис. 3.3).

Рисунок 3.3

ВИСНОВОК

Протягом написання цієї роботи я навчився, як працювати з Unity, познайомився з великою кількістю цікавих елементів, механік та функцій. Таких як: контроль камери, налаштування води через скрипт, особливості механіки переміщення по карті та взаємодія з камерою, як переносити в Unity об’єкти з унікальним стилем, вивчив тонкощі створенню локацій та їх налаштування, як на FPS впливає моє перенесення об’єктів на мапу, як можна полегшити власну розробка завдяки інноваційним технологіям (ШІ), навчився розробляти повноцінний UI, як правильно працювати, прикріпляти анімації персонажів та звідки, якщо треба, можна брати 3D моделі для створення ігор, навіть не витрачаючи на це кошти.

Завдяки цьому, зрозумів, чому саме Unity, вважають кращою програмою для написання невеликих за розміром ігор. Також це дозволило мені по іншому подивитись на розробку таких проєктів як цей. Відчути та зрозуміти, для чого так необхідно експериментувати в написанні коду. Чому програмісти повинні дивитись на розробку, скоріше як на одну велику головоломку, а ніж задачу, що треба виконати в срок.

Я дуже радий що обрав для написання, легенду про «Гамельнського Щуролова», бо сам завжди був фанатом легенд та міфів, а написання цієї роботи, показало мені що навіть як розробник, я можу стати ближчим до таких історій через написання діалогів, коду або створенню локацій. В якомусь сенсі це можна навіть назвати «переосмисленням» того, яка ідея лежить в основі, для чого усе закінчилось так, як це вийшло в історії.

Це був мій перший досвід, а багато елементів, що були використані в цій роботі, були опановані мною з повного нуля, тому отриманий результат приносить ще більше насолоди та бажання розвиватись у цьому напрямку.

Завдяки розуміння усього що відбувається на екрані з точки зору програмування, починаєш розуміти, що оптимізація – це не лише технічний етап, а й творчий процес, де важливо знаходити баланс між красотою та продуктивністю. Найголовніше – забезпечити комфортний ігровий досвід,

навіть якщо для цього доводиться йти на певні жертви у графіці або витратити більше часу на знаходження більш оптимального варіанту.

Я дякую, за можливість попрацювати в такому напрямку, хоча і не так довго як хотілося б, але сподіваюсь що одного разу я повернусь до розробки цієї гри та доведу її до рівня не просто «проєкту», а повноцінної гри, яку буде nm не соромно виставити на таких ігрових платформах як Steam та Epic Games.

Список використаних джерел

1) Перемот, О. Команда Unity запустила AI Marketplace та показала перші верифіковані рішення на базі штучного інтелекту для розробки ігор. DOU. URL: https://gamedev.dou.ua/news/unity-ai-asset-store/

2) Основи движка Unity 2025 з нуля - як створити гру на Юніті, курс розробки на itProger – курси українською. itProger. URL: https://itproger.com/ua/course/unity

3) Гаранін, О., Мосієнко, Н. Адаптивний штучний інтелект у RPG-грі на ігровому рушії Unity. URL: http://elibrary.kdpu.edu.ua/xmlui/handle/123456789/2891

4) Оксенюк, В., Юрчак, І., Маркелов, О., Шурко, М., Рубаха, М., Черниш, А. АРХІТЕКТУРА ТА РЕАЛІЗАЦІЯ МОБІЛЬНОГО ЗАСТОСУНКУ CALORIFY НА ОСНОВІ РУШІЯ UNITY. DSpace at KDPU. URL: https://sciencetst.lpnu.ua/uk/cds/vsi-vypusky/volume-6-number-2-2024/arhitektura-ta-realizaciya-mobilnogo-zastosunku-calorify-na

5) Калинич, Ю., Білак, Ю., Небесний, Р., Федорка, П. Аналіз процесів формування симуляцій з використанням графічного процесора. DSpace at KDPU. URL: https://science.lpnu.ua/uk/sisn/vsi-vypusky/vypusk-11-2022/analiz-procesiv-formuvannya-symulyaciy-z-vykorystannyam-grafichnogo

6) Глуховський, М. DGlibrary. Методи оптимізації ігрового ПЗ засобами Unity. URL: https://dglibtest.nubip.edu.ua/items/81cb9f87-07f4-43d8-80e1-3b639aea56ae

7) Ростальний, Б. Моісеєнко, Н. Розробка гри-квесту засобами рушія Unity. DSpace at KDPU. URL: https://elibrary.kdpu.edu.ua/handle/123456789/5382

8) Компанець, А., Чемериш, Г., Крашенинник, І. Using 3D modelling in design training simulator with augmented reality. DSpace at KDPU. URL: https://elibrary.kdpu.edu.ua/handle/123456789/3740

9) Кузнецов, В., Моісеєнко, Н., Ростальний, М. Using Unity to Teach Game Development. Using Unity to Teach Game Development. DSpace at KDPU. URL: https://elibrary.kdpu.edu.ua/handle/123456789/7034

10) Saver, M. Games made with Unity: April 2025 in review. Unity. URL: https://unity.com/blog/games-made-with-unity-april-2025-releases

11) Супрун, Б., Створення 3d гри з використанням алгоритму пошуку в середовищі Unity. Чорноморський національний університет імені Петра Могили. URL: https://krs.chmnu.edu.ua/jspui/bitstream/123456789/2392/1/402%20%D0%A1%D1%83%D0%BF%D1%80%D1%83%D0%BD%20%D0%91%D0%BE%D0%B3%D0%B4%D0%B0%D0%BD%20%D0%9E%D0%BB%D0%B5%D0%BA%D1%81%D0%B0%D0%BD%D0%B4%D1%80%D0%BE%D0%B2%D0%B8%D1%87.pdf

12) Самко, М., Що таке Unity? Lemon. URL: https://lemon.school/blog/shho-take-unity

13) Красніков, Д., Unity URP. Туторіал для початківців. DOU. URL: https://gamedev.dou.ua/forums/topic/42928/

14) SC блог. Підручник головного меню для Unity. Sharp Coder Blog. URL: https://uk.sharpcoderblog.com/blog/unity-3d-create-main-menu-with-ui-canvas

15) SC блог. Робота з системою інтерфейсу Unity. Sharp Coder Blog. URL: https://uk.sharpcoderblog.com/blog/working-with-unitys-ui-system

ДОДАТКИ

Додаток А: Приклад коду

```
using UnityEngine;
using System.Collections;

public class RatFollower : MonoBehaviour
{
    public bool isControlled = false;
    public Transform target;
    public float followSpeed = 3f;
    public float rotationSpeed = 5f;
    public float stoppingDistance = 1.5f;
    public float wanderRadius = 5f;
    private float wanderSpeed;
    public float minWanderTime = 1f;
    public float maxWanderTime = 3f;
    private Vector3 currentWanderCenter;
    private Vector3 wanderTarget;
    private bool isWandering = false;
    private void Start()
    {
        SetNewWanderCenter(transform.position);
        wanderSpeed = followSpeed * 2f;
    }
    private void Update()
    {
        if (isControlled)
        {
            FollowTarget();
        }
        else
        {
            if (!isWandering)
            {
                StartCoroutine(Wander());
            }
            MoveToWanderTarget();
        }
    }
    public void SetControlled(bool controlled)
    {
        if (isControlled && !controlled)
        {
            SetNewWanderCenter(transform.position);
        }
        isControlled = controlled;
    }
    private void SetNewWanderCenter(Vector3 newCenter)
```

```
    {
        currentWanderCenter = newCenter;
        wanderTarget = GetRandomWanderTarget();
    }
    private void FollowTarget()
    {
        if (target == null)
            return;

        Vector3 direction = target.position - transform.position;
        direction.y = 0;

        if (direction.magnitude > stoppingDistance)
        {
            transform.position = Vector3.MoveTowards(
                transform.position,
                target.position,
                followSpeed * Time.deltaTime
            );
        }

        if (direction.magnitude > 0.1f)
        {
            Quaternion targetRotation = Quaternion.LookRotation(direction) * Quaternion.Euler(-90,
180, 0);
            transform.rotation = Quaternion.Slerp(
                transform.rotation,
                targetRotation,
                rotationSpeed * Time.deltaTime
            );
        }
    }
    private IEnumerator Wander()
    {
        isWandering = true;
        wanderTarget = GetRandomWanderTarget();

        float waitTime = Random.Range(minWanderTime, maxWanderTime);
        yield return new WaitForSeconds(waitTime);

        isWandering = false;
    }
    private void MoveToWanderTarget()
    {
        Vector3 direction = wanderTarget - transform.position;
        direction.y = 0;

        if (direction.magnitude > 0.5f)
        {
            transform.position = Vector3.MoveTowards(
```

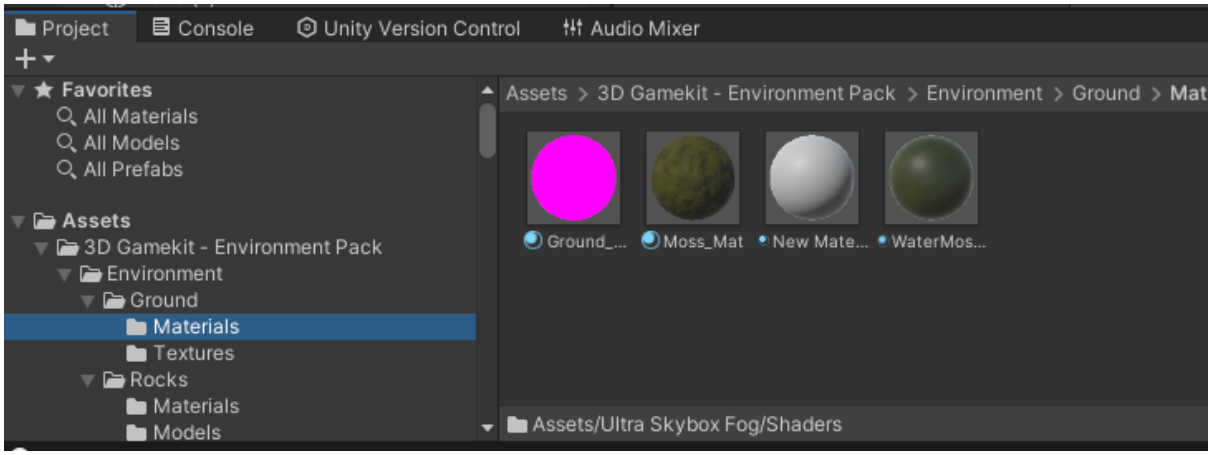
```
            transform.position,
            wanderTarget,
            wanderSpeed * Time.deltaTime
        );
        if (direction.magnitude > 0.1f)
        {
            Quaternion targetRotation = Quaternion.LookRotation(direction) * Quaternion.Euler(-90,
180, 0);
            transform.rotation = Quaternion.Slerp(
                transform.rotation,
                targetRotation,
                rotationSpeed * Time.deltaTime
            );
        }
    }
  }
  private Vector3 GetRandomWanderTarget()
  {
    Vector2 randomCircle = Random.insideUnitCircle * wanderRadius;
    Vector3 randomPoint = currentWanderCenter + new Vector3(randomCircle.x, 0,
randomCircle.y);

    if (Vector3.Distance(currentWanderCenter, randomPoint) > wanderRadius)
    {
      randomPoint = currentWanderCenter + (randomPoint - currentWanderCenter).normalized *
wanderRadius;
    }

    return randomPoint;
  }
}
```

Додаток Б: Інші приклади в якості скріншотів

Додаток Б 1 (Приклад проблеми відображення деяких матеріалів):



Додаток Б 2:

Приклад роботи системи контролю щурів з відстані

Додаток Б 3:

Приклад додаткових функцій Terrain в Unity (створення дірок в поверхні, завдяки чому можна створювати справжні печери або тунелі)